Robotics Architectures for Indoor and Outdoor Museum Guides

Robotic Architectures for indoor and outdoor museum guides

Lorenzo Riano

February 14, 2008

Contents

List of Figures

Acknowledgments

Dal primo momento in cui ho iniziato a scrivere questa tesi ho pensato a come scrivere i ringraziamenti. Ci sono persone che hanno direttamente partecipato alla sua stesura, e persone che mi hanno sostenuto tantissimo pur non sapendo cosa vuol dire quello che ho scritto. Cercando di rendere omaggio alla terra che mi ha ospitato durante il dottorato, ho preso una semplice decisione: non inserisco nomi, né riferimenti. Chiunque ci tenesse a vedere comparire il suo nome come preambolo ad una lista di equazioni, immagini picassiane e termini strani sarà deluso. Ma sono sicuro che chiunque sa di essermi stato vicino, di avermi aiutato o anche solo stimolato nelle mie ricerche, troverà un ringraziamento lungo circa cento pagine di tesi.

Ma non posso cavarmela così facilmente. Esistono persone che mi hanno sopportato, aiutato, che mi hanno confortato, e spesso riportato ai miei doveri quando non ce la facevo più. Senza di loro tutto, sarebbe diverso oggi. Senza di loro, penso che buona parte di questo lavoro non avrebbe la forma che ha oggi. Eppure hanno fatto una cosa così semplice: mi hanno dato uno scopo. Ed è a loro che dedico le righe più importanti di questa tesi.

Grazie.

Part I

Overview

Chapter 1

Introduction

The field of museum robotic tour-guide is a real testbed for applied robotics. Algorithms for mapping, localization, navigation and human-robot interaction are deployed in an out-of-lab environment and integrated in a robotic system to serve people. Usually the tour-guide robots environments are highly dynamic and unpredictable, as hundreds of people walk around and make the robot job harder and harder. A good robotic architecture should cope with any kind of uncertainty, trying to react to unpredictable fault conditions, while still maintaining a pleasant behavior for the museum visitors.

Consider for example the task of a human guide whose goal is to show the museum and its exhibits to a group of visitors. The first thing he should know is a list of the museum exhibits, together with their positions in a suitable frame of reference. For example he could know that in a museum room there are three main exhibits, while in the next two rooms there are items of minor interest. He should know therefore where the rooms are inside the museum, where the exhibits are located in each room and he should plan a tour to visit them. Obviously a human guide who stops in the middle of a room trying to remember where a place of interest is does not look very amazing. The same problem is even harder if we consider a large unstructured environment, like an archaeological garden. Here distances play a fundamental role, as an incorrect tour planning could lead to the guide (and the following visitors) walking for kilometers to see exhibits very far away from each other.

Now consider the same problem, but applied to a robotic platform. While we human being consider the task of walking as somewhat innate, thus very easy, the same is not true for a robot. Tasks like moving from one place to another one while not colliding with obstacles are trivial for us, while the robot needs ad hoc algorithms that should often include a detailed kinematic model of the robot locomotion system.Many algorithms for obstacle avoidance have been developed, but they all relies on the robot sensing capabilities, which are often very limited.

For example a glass door is hard to detect for most of the sensors in commerce (and often for human too), and the robot could easily collide with it. Another "hard problem" for the robotics is the world state estimation. Most of the robotic algorithms rely on a complete world state knowledge, which is hard to achieve even for a very simplified world model. An instance of this problem is tracking the robot position: no sensor can estimate with precision the robot pose in a given frame of reference, and if we consider measure uncertainty over the poses space, the problem quickly grows to become intractable. The same problem is magnified when considering large and unstructured environments. Consider yourself in the middle of an unknown city, without a map, trying to get to a particular place with nobody to ask for information. You could move around and explore your neighborhood, but after traveling along roads and turning at crossings it is hard to decide if the place you are looking at is the same you were some time ago. Even if you have a map but you do not know your starting place, it could take a lot of time and exploration efforts to understand where you are. A similar problem arises when your map is not detailed enough, as it often happens for touristic maps.

In the last decade we saw many successful robots deployed in public museums and acting as tour-guides (some of them will be introduced in section 1.2). Issues like navigation, planning and map-building have been mostly addressed. The key idea in successful robotic applications is to explicitly introduce uncertainty in each algorithm. Uncertainty in robotics arises from the sensors, from the actuators, and from the world representation model. It is not possible, apart from trivial scenarios, to build a robust robotic architecture without coping with uncertainty. Although this has been observed well before the last decade, it is only in the near past that robotic architectures have been pervaded with uncertainty measures.

Uncertainty means stochastic. We do not have anymore a single variable describing the world state, but a probability distribution over all the world states space. We do not have anymore a deterministic action, but actions are functions over a probabilistic space. The same is true for decisions: the robot does not take decisions based on sure outcomes, but on an expected utility based on the robot beliefs. The probabilistic approach has a drawback: algorithms are often computationally expensive, at the point that they are intractable. The problem of finding a good trade-off between uncertainty representation and computation is still mostly unresolved.

This thesis is mainly concerned with robotic architectures for museum tour-guides. We will consider two different environments, namely the Archaeological Museum of Agrigento and the Botanical Garden of Palermo. Each developed architecture will try to solve a set of problems, while pursuing the goal of creating a fault tolerant robotic system. The robot Cicerobot operates in the archaeological museum, and it uses well-known algorithms for mapping, localization and planning processes. Although it is able to fulfill its tasks, it lacks a high level control

system to recover after faults or to take decisions when multiple action choices are presented. To to this aim we will develop a high level decision system based on dynamic Bayesian networks and influence diagrams. These represent a graphical language to aid a decision system taking decision under uncertainty. The robot may decide, for example, to ask people clear its way or to try a maneuver to avoid them, depending on a measure on the number of people surrounding it and the expected outcomes of its strategy. Furthermore the dynamic Bayesian network is used to estimate some unknown variable using information from many sensors and its belief. Therefore the proposed robotic architecture will be more fault tolerant, and even more appealing to people.

While Cicerobot operates in an indoor environment, Robotanic operates in a large outdoor environment, namely a botanical garden. Dealing with outdoor environments is harder than dealing with indoor ones, as spaces are very large and sensor data are often very subject to noise. Furthermore outdoor environments are often unstructured, meaning that they lacks many features useful for localization and navigation like walls or uniform lighting. During our experiments we found that a metric representation is not suitable for this environment (in contrast with indoor ones). We will explore two ways to represent the environment, namely appearance and topological maps.

Appearance based navigation uses a *holistic* view of image data to memorize the "look-like" of a place. Instead on focusing on landmark detection or features detection, the holistic approach uses all the information gathered from a set of images to describe a place. We will develop a robotic system which fuses camera and odometric information to autonomously partition the environment into places. Next they are connected in a graph to allow planning and navigation. This approach lifts the need of estimating the robot position point-wise, while still allowing precise localization and navigation.

The last developed architecture uses a functional definition of meaningful place in order to build a topological map of the environment. A topological map is a graph structure in which meaningful places are represented by nodes, and the link between nodes implies a physical connection between places. The graph is built by incrementally adding or removing links as the beliefs are updated using a dynamic Bayesian Network. Furthermore 3D vision is used in order to detect branching in the pathways and to estimate the free space in front of the robot.

Several public demos validated the capabilities of the proposed robot architectures to deal with issues related to perception, planning and human-robot interaction typical of museum tour applications. Cicerobot has been tested during a first session of experiments in the period from March to July 2006. The second session, during which we used the high-level decision system (chapter 4), started in November 2007 and ended in January 2008. Robotanic has been tested in the period from April to September 2007 and it has been presented during an inter-

national conference held in the Botanical Garden of Palermo.

The main contribution of this thesis are i) the use of an influence diagram for high level robot controlling, an approach that is, as far as we know, still unexplored; ii) the study of non-metric environment representations for large unstructured environments, allowing the robot to reliably navigate and localize even when precise position information are not available.

In the next two sections we will give a brief outline of this thesis and a survey of the museum tour-guide literature.

1.1 Outline

This thesis is organized in four parts. Each part is divided in chapters, in which we try to address some issues that often arose in the previous ones. Each chapter is meant to be self-explanatory, thus providing an introduction to the problem, related works, system development and conclusions.

In chapter 2 we will introduce the localization and mapping problem, which is one of the main topics in robotic. We will introduce furthermore planning and control, illustrating the approaches we will use in the development of the proposed architectures.

The second part introduces a robotic architecture for indoor robotic museum guide, based on the concepts presented in chapter 2. It is divided in two chapters: chapter 3 illustrates the development of the robot "Cicerobot", operating at the Agrigento Archaeological Museum. This work is very similar to the ones in literature, but the issues that will arise in this environment will be the motivation for the development of a high level control architecture presented in chapter 4. Such architecture is based on Bayesian networks, a framework for reasoning under uncertainty, with the aim of providing a fault tolerant robotic controller. It should pointed out that the proposed controller is not concerned with issuing low level commands, but only to reason about the robot state and act accordingly.

The third part is concerned with the development of an outdoor robotic museum guide architecture, which is the main focus of this thesis. As the environment and the robotic platform are much more different from the previous ones, we had to develop a complete new system, illustrated in chapter 5. Following this experience we tried to answer a hard question: "Does the robot need a complete metric representation of the environment?". The answer obviously depends on the task the robot should carry on, but in chapter 5 we will see that a coarse topological representation augmented with local-scale metric information is sufficient to solve the proposed task. In the two subsequent chapters we will develop two robotic systems aimed to build a topological map of the environment without any metric information. In chapter 6 we will explore the appearance based robot mapping

and localization, adapting current state-of-the-art algorithms to the problem of on-line learning. Although the architecture will not be used in the context of robotic museum guides, it will form the basis for the development of the system proposed in chapter 7. According to the proposed system the robot should be able to build a representation of the environment based on junction roads, obtaining a map similar to an urban one. As a mean to explore the visual SLAM approach, we will use only camera information.

The last part presents a conclusions and still open issues.

1.2 State of the art

In this section we will provide a brief survey of the robotic museum guides topic. We will outline the most relevant works from the adopted techniques point of view. Furthermore each chapter includes a more detailed and focused related works section, as demanded by the chapter main topic.

One of the first tour-guide robot was Polly [1], which operated at the AI Lab of MIT. It used a camera-detected floor carpet as a pathway for movement and obstacles detection, and a behavior-based architecture for control. Although being a very simple robotic architecture, it could be considered the pioneer of robotic tour-guides. Rhino [2, 3] has probably been the first robotic museum guide. It operated in the Deutsches Museum in Bonn, and it used most of the algorithms we will describe in chapter 2, although most of them were not as refined as today's standards. Paralleling the development of Rhino the tour-guide robot Xavier [4] has been deployed at the Carnegie Mellon University. People could use a Web interface to give it commands, which were mostly navigation-related. Its localization and navigation system was based on *Partially Observable Markov Decision Process* [5]. This is a generalization of the *Markov Decision Process* (MDP) where the world state is not observable, but only detectable through sensors. The robot pose is tracked using a discrete Bayes filter, where the state space is discretized to a coarse grid. POMDP are used to plan the robot actions. As computing a policy for a POMDP may be a very expensive task, all the policies are computed offline and then used when needed.

A masterpiece in the topic of robotic museum guides is Minerva [6]. Much of the chapter 2 is based on this work. Minerva successfully operated for two weeks at the Smithsonian Museum. It used an EM algorithm to generate an environment map, and Monte Carlo Localization to track the robot pose. POMDP were employed to generate a plan for navigation; the result is also known as *coastal navigation*, referring to the sailor habit of navigating near the coasts to avoid getting lost. A similar concept is applicable to robotics, where a robot should prefer a path close to mapped objects in order to refine its belief about its position.

Furthermore an emotional system has been developed in order to facilitate human-robot interactions.

A different approach is in [7]. They deployed a tour-guide robot named RobotX at the Swiss National Exhibit Expo. It used a multi-hypothesis Gaussian localization system to track the robot position, while the environment was represented using a topological graph. Sensor data are integrated by matching local features with global ones, organized in a search tree. Navigation and planning are performed by a Navigation Function mixed with a Dynamic Window Controller (see section 3.5.

Finally, a simple but very effective robotic architecture has been used to control Sage [8], a robot operating at the Carnegie Museum of Natural History. It used a behavior based architecture to control its movements, and a set of artificial landmarks to track its position. Although modifying the environment is not always possible, the authors pointed out that a simple architecture and a simple implementation made a robot reliable even when operating in fully autonomous mode.

As far as we know no outdoor robotic museum tour-guide has been developed so far. In the chapters devoted to outdoor robotic we will give some references about some related works and the outdoor robotic topic in general.

Chapter 2

Probabilistic Robotics

Signor capitano, non glielo volevo dire
Ma in mezzo al mare c'è una donna bianca
[...]
Giovanotto, io non vedo niente
Andiamo avanti tranquillamente
- Francesco de Gregori, I muscoli del capitano

One of the major challenges of modern robotics is estimating the robot and the world state. Most of the algorithms developed in the field of robotics rely on a somewhat precise observation of the robot state and the world geometry. For example most planning algorithms need to know the precise robot position together with a detailed map of its environment. Even if the robot could have a "perfect" sensor, i.e. a sensor which is able to measure the robot and the world state with absolute precision, dynamics and changes in the environment make each plan useless after just a little time. Nature makes the task even harder, as any action or perception is every time corrupted by noise, which often accumulates with time.

These considerations lead to the conclusion that dealing with the real world using physical devices cannot be accomplished, apart from trivial tasks, without explicitly considering uncertainty. Probabilistic robotics is the field of applied robotics where uncertainty in sensing and acting are treated using statistical algorithms. The key idea behind probabilistic robotics is that instead of representing the robot state as a single deterministic variable, it is represented as a distribution function over the whole state space. The world too is represented using a measure of uncertainty, which comprises either errors in model building, either changes due to environment modifications. Moreover even actions and perceptions are modeled by considering the intrinsic noise they are affected. For example a modern laser rangefinder has an average error of about $10cm^2$, even if there are some surfaces which produce strange light reflections or even no reflection at all. Sonars are much more affected by noise, as they measure distance in a cone instead of a beam, so

that we cannot know which part of the cone generated the reflection. One of the best known errors that plague mobile robotics concerns odometry. Wheels slippage or even small errors in the odometric model accumulate over time, making the estimated robot pose completely unknown after a few movements. Other common sources of uncertainty are from GPS data or digital compasses.

In this chapter we will provide an overview of common approaches to probabilistic robotics, mainly illustrating the algorithms we will use in the following chapters. We refer to [9] for a complete survey of the topic.

2.1 State estimation

2.1.1 Notation

One of the main goals of robotics is to estimate the environment state x_t at time t. The degree of information of the variable x depends on the current task: often it will be the robot pose in a given frame of reference, but it may include external sources of information like landmarks or people position. We will assume that the state variables undertake the *Markov Assumption*, i.e.:

$$p(x_{t+1}|x_t, x_{t-1}) \equiv p(x_{t+1}|x_t) \tag{2.1}$$

This assumption is often violated when dealing with environment dynamics (e.g. people) not explicitly included in the state variable, or even if some erroneous assumptions are included in the models. We will see that the Bayes Filter described below is often robust to Markov Assumption violations. We will refer to the environment state distribution as *robot belief* or simply the *belief* $bel(x_t)$.

A robot is a physical device which interacts with the environment. Interaction is performed by *acting*, e.g. moving, and *sensing*. An action u_t usually modifies the environment, in the sense that the state variable x is modified after taking an action. This means that we are often interested in the probability distribution $p(x_t|u_t, x_{t-1})$, i.e. the probability of being in state x_t after applying the action u_t conditioned on the robot being previously in state x_{t-1}. We will refer to this distribution as *motion model*, but sometimes it is called *transition probability* or *predictive distribution*.

Sensing is the act of observing the environment state using the robot sensors. This is a special kind of action in the sense that it does not modify the environment, but may give some insights in the current state. In other words, a sensing z_t may modify the current belief x_t according to the distribution $p(x_t|z_t)$. Usually estimating this distribution is very hard: imagine estimating the robot pose x_t

given a camera image z_t!. We can invert the variables using the Bayes rule:

$$p(x_t|z_t) = \frac{p(z_t|x_t)p(x_t)}{p(z_t)} \tag{2.2}$$

The distribution $p(z_t|x_t)$, that is called the *sensor model*, is often much simpler to compute, as we will see in section 2.3. Although we could use the Markov Assumption for the distribution $p(x_t|z_t)$, using past readings like in $p(x_t|z_{1:t})$ usually yields better results (for example in scan matching iterations are performed over all the dataset). The converse is not true, as the current readings are independent of past data given the robot belief:

$$p(z_t|x_{1:t}, z_{1:t-1}) \equiv p(z_t|x_t) \tag{2.3}$$

Finally we define a static description of the environment a map m. We use the term "static" here to underlie that any action performed by the robot does not change the map, so it is not included in the state variable x_t. Maps may be given to the robot, like a building blueprint, or they may be learned using mapping algorithms (section 2.6). Either the sensor model and the motion model may be conditioned on the map m. Common representations of maps include metric grid-like maps, like *occupancy grids*, or features map, like a set of landmarks and their poses.

2.1.2 Belief Updating

The robot belief at time t, conditioned on past readings and actions, is represented by the distribution $p(x_t|z_{1:t}, u_{1:t})$. We will show that it can be computed using a recursive formula, i.e. by building on the past belief $p(x_{t-1})$. The first step is the application of the Bayes rule:

$$p(x_t|z_{1:t}, u_{1:t}) = \frac{p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t})}{p(z_t|z_{1:t-1}, u_{1:t})} \tag{2.4}$$

The denominator is just a normalizing constant, and we will refer to it by the variable η. By applying the Markov Assumption we get:

$$p(z_t|x_t, z_{1:t-1}, u_{1:t}) = p(z_t|x_t) \tag{2.5}$$

The second term in equation 2.4 is reduced by marginalization over x_{t-1} and by further application of Markov Assumption:

$$\begin{aligned} p(x_t|z_{1:t-1}, u_{1:t}) &= \\ &= \int p(x_t|z_{1:t-1}, u_{1:t}, x_{t-1})p(x_{t-1}|z_{1:t-1}, u_{1:t})dxt - 1 \textit{ by marginalization} \\ &= \int p(x_t|u_t, x_{t-1})p(x_{t-1}|z_{1:t-1}, u_{1:t-1})dxt - 1 \textit{ by Markov} \end{aligned} \tag{2.6}$$

The term $p(x_{t-1}|z_{1:t-1}, u_{1:t-1})$ is the belief at time $t-1$, so we can rewrite equation 2.4 using the **Bayes filter** recursive formula:

$$bel(x_t) = \eta p(z_t|x_t) \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1} \tag{2.7}$$

In equation 2.7 we can recognize two already introduced terms: the motion model and the sensor model. We can describe the Bayes filter by saying that, starting with the belief at time $t-1$, we first update it using the motion model. This step is called *prediction step*, as it computes the robot state using an open loop prediction step. The next step is to integrate observations by using the sensor model, which narrows the belief around the most probable state given actions and observations.

Although the Bayes filter represents an elegant solution to the belief updating problem, it is usually not computable in closed form[1]. We should rely on approximate solutions, by making assumption on the belief distribution shape or by discretizing it. A common approach is the *Kalman filter* [10]; it is an optimum estimator under the assumption that the belief function is a Gaussian distribution, and the assumption that booth sensor and motion models are linear functions of the belief. The linear functions assumption has been partially lifted by using the *Extended Kalman filter* or the *Unscented Kalman filter* [11], while the Gaussian assumption still limits the filter applications.

A popular method that gained much attention in the last years is the *Particle Filter* which traces back to the work by Metropolis and Ulam [12]. According to this approach the belief is represented as a finite collection of samples, named *particles*. By using a discrete set of samples we can replace the integral in equation 2.7 with a summation, making the belief update formula computable given enough computational resources. In the next section we will describe the Particle filter, as it will be heavily used in the next chapters.

2.2 Particle Filter

If we could have a large set of samples from the belief distribution, we could easily solve equation 2.7. This is the basic insight of Monte Carlo methods, which use a discrete representation of the target posterior distribution in order to approximate integrals. Let us assume that we are able to draw N independent and identically distributed (i.i.d) random samples $\{x_{0:t}^{(i)}; i = 1 \dots N\}$ from the belief $bel(x)$. Then

[1]The main problem lies not only in the normalizing factor η, but also in the second term integral in equation 2.7.

we could approximate it by the distribution:

$$\widetilde{bel}(x) = \frac{1}{N}\sum_{i=1}^{N}\delta(x_{0:t}^{(i)}) \tag{2.8}$$

where $\delta(x_{0:t}^{(i)})$ is the delta-Dirac mass located in $x_{0:t}^{(i)}$. The filter equation 2.7 then becomes:

$$\begin{aligned}\widetilde{bel}_t(x) &= \\ &= p(z_t|x_{0:t}^{(i)})\int p(x_{0:t}^{(i)}|u_t, x_{0:t-1}^{(i)})\widetilde{bel}_{t-1}(x) \\ &= p(z_t|x_{0:t}^{(i)})\frac{1}{N}\sum_{i=1}^{N} p(x_{0:t}^{(i)}|u_t, x_{0:t-1}^{(i)})\end{aligned} \tag{2.9}$$

The main issue related to this approach is that we cannot generate i.i.d samples from an arbitrary multi-variate distribution. We need an alternative sampling methods, known as the *Importance Sampling*. Suppose that we need to evaluate a distribution $P(x)$, but we can only evaluate it up to a multiplicative constant. In other words we can evaluate a function $P^*(x)$ such that

$$P(x) = \frac{P^*(x)}{Z} \tag{2.10}$$

where Z is, in our case, the normalizing constant η. Let assume furthermore that we have a simpler distribution $Q(x)$ which we can evaluate again up to a multiplicative constant (i.e. $Q(x) = Q^*(x)/R$). We will call Q the *proposal distribution* and we will assume that we can draw N samples $\{x_{0:t}^{(i)}\}$ from it. As these samples are drawn from a different distribution $Q^*(x)$, we can account for the mismatch between the two distributions by introducing the *weight*

$$w_t^{(i)} = \frac{P^*(x_t^{(i)})}{P^*(x_t^{(i)})} \tag{2.11}$$

Equation 2.8 then becomes:

$$\widetilde{bel}(x_t) = \frac{\sum_{i=1}^{N} w_t^{(i)}\delta(x_t^{(i)})}{\sum_{i=1}^{N} w_t^{(i)}} \tag{2.12}$$

If $Q(x)$ is non-zero for all x where $P(x)$ is non-zero, then it could be proved that the approximate belief converges to the true one as the number of particles N goes toward infinity.

The next step is to describe the procedure described above using the distributions we already described, namely the motion model and the sensor model. By replaying the steps in equations 2.4 and 2.6, this time maintaining all states in the posterior, we get:

$$p(x_{0:t}|z_{1:t}, u_{1:t}) = \eta p(z_t|x_t)p(x|x_{t-1}, u_t)p(x_{0:t-1}|z_{1:t-1}, u_{1:t-1}) \tag{2.13}$$

If we use the motion model $p(x|x_{t-1}, u_t)$ as a proposal distribution, we get the weight:

$$\begin{aligned} w_t &= \\ &= \eta \frac{p(z_t|x_t)p(x|x_{t-1}, u_t)p(x_{0:t-1}|z_{1:t-1}, u_{1:t-1})}{p(x|x_{t-1}, u_t)p(x_{0:t-1}|z_{1:t-1}, u_{1:t-1})} \\ &= \eta p(z_t|x_t) \end{aligned} \tag{2.14}$$

where the term η is irrelevant because sampling is performed using an alternative distribution $Q^*(x)$. The same procedure is repeated for each particle, leading to the Particle filter algorithm:

Algorithm 1 Particle Filter($u_{1:t}$,$z_{1:t}$)

1: **for** $i = 1, \ldots, N$ **do**
2: sample $x_0^{(i)} \sim p(x_0)$
3: **end for**
4: **for all** t **do**
5: **for** $i = 1, \ldots, N$ **do**
6: sample $\tilde{x}_t^{(i)} \sim p(x_t|x_{t-1}, u_t)$
7: $\tilde{w}_t^{(i)} = p(z_t|x_t)$
8: **end for**
9: Normalize the particles weights
10: Resample with replacement N particles $\{x_t^{(i)}\}$ from the set $\{\tilde{x}_t^{(i)}\}$ according to the importance weights and append them to the belief $bel(x_t)$
11: **end for**

This algorithm, that is also known as Bootstrap filter [13], is very easy to implement and generalizes to a very broad range of applications. It will be the main localization algorithm we will use in the next chapters. For further details see [14, 15].

2.3 Sensor Model

The sensor model $p(z_t|x_t)$ is used to estimate the probability of a percept z_t given the robot position x_t. Usually we will assume that an environment map m is

available, so that the sensor model become $p(z_t|x_t) \equiv p(z_t|x_t, m)$. In this section we will introduce the *beam model* [16, 17], a sensor model designed for accurate range sensors like lasers. According to this model, the resulting distribution is a mixture of four distributions:

- **Real object with local noise**: A range finder may correctly hit the nearest object as reported in the map. Let the true nearest object in the laser trajectory at angle θ be denoted by $z_t^{\theta*}$. This may be easily computed by ray-casting a beam from the robot in position x_t in the map m and at angle θ until it hits an obstacle. In the real world the laser is affected by noise, assumed to be estimable by a Gaussian distribution with variance σ_{hit} (usually the laser manufacturer specifies the noise variance). Thus we model the hit probability as:

$$p_{hit}(z_t^\theta|x_t, m) = \begin{cases} \eta\mathcal{N}(z_t^\theta, z_t^{*\theta}, \sigma_{hit}) & \text{if } 0 \leq z_t^\theta < z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

  where z_{max} is the maximum range of the laser and $\mathcal{N}(x, \mu, \sigma)$ is a Gaussian distribution with mean μ and variance σ. The normalizer η evaluates to:

$$\eta = \left(\int_0^{z_{max}} \mathcal{N}(z_t^\theta, z_t^{*\theta}, \sigma_{hit}) dz_t^\theta \right)^{-1} \tag{2.16}$$

- **Short readings**: Unmodeled environment dynamics are explained by an exponential distribution. Typical moving objects are people: they tend to be attracted by the robot often surrounding it (see section 3.6 for an example and applications). The required distribution is:

$$p_{short}(z_t^\theta|x_t, m) = \begin{cases} \eta\lambda_{short} e^{-\lambda_{short} z_t^\theta} & \text{if } 0 \leq z_t^\theta < z_t^{\theta*} \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

  where the normalizer η evaluates to:

$$\eta = \left(\int_0^{z_t^{\theta*}} \lambda_{short} e^{-\lambda_{short} z_t^\theta} dz_t^\theta \right)^{-1} = \frac{1}{1 - e^{\lambda_{short} z_t^{\theta*}}} \tag{2.18}$$

- **Max range**: Usually a range finder returns z_{max} if it did not report an obstacle in range. This is modeled by a point mass distribution centered in z_{max}:

$$p_{max}(z_t^\theta|x_t, m) = \begin{cases} 1 & \text{if } z_t^\theta = z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{2.19}$$

- **Unmodeled random noise**: Further unexpected laser reflections are equiprobable over all the laser ranges:

$$p_{rand}(z_t^\theta|x_t, m) = \begin{cases} \frac{1}{z_{max}} & \text{if } 0 \leq z_t^\theta < z_t^{\theta *} \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

The sensor model for a single beam $p(z_t^\theta|x_t, m)$ is a linear combination of the four distributions described above, using coefficients z_h, z_s, z_m, z_r which sum to 1:

$$\begin{pmatrix} z_h \\ z_s \\ z_m \\ z_r \end{pmatrix}^T \cdot \begin{pmatrix} p_{hit} \\ p_{short} \\ p_{max} \\ p_{rand} \end{pmatrix} \tag{2.21}$$

Finally we assume that each measure is independent of the others, thus obtaining the final sensor model:

$$p(z_t|x_t, m) = \prod_\theta p(z_t^\theta|x_t, m) \tag{2.22}$$

2.4 Motion Model

The prediction step in the Bayes filter is performed by the motion model $p(x_t|u_t x_{t-1})$. It computes the probability of being in a state x_t after applying control u_t and while previously being in state x_{t-1}. Here the control is assumed to be the traslational and rotational velocities v, ω. We will use the particle filter described in 2.2, so we need to generate samples from the motion model. This is performed by firstly perturbing the control using random noise, then applying the perturbed control to the initial state x_{t-1}.

The first step is to apply random noise to the control:

$$\begin{pmatrix} \hat{v} \\ \hat{\omega} \\ \hat{\gamma} \end{pmatrix} = \begin{pmatrix} v \\ \omega \\ 0 \end{pmatrix} + \begin{pmatrix} \mathcal{N}(0, a1|v| + a2|\omega|) \\ \mathcal{N}(0, a3|v| + a4|\omega|) \\ \mathcal{N}(0, a5|v| + a6|\omega|) \end{pmatrix} \tag{2.23}$$

where $\mathcal{N}(0, \sigma)$ is a random sample generated from a Gaussian distribution with mean 0 and variance σ. The next step is to apply the kinematics equations of motion:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} \hat{x} \\ \hat{y} \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{w}\sin(\theta) + \frac{v}{w}\sin(\theta + \omega\Delta t) \\ \frac{v}{w}\cos(\theta) + \frac{v}{w}\cos(\theta + \omega\Delta t) \\ \omega\Delta t + \hat{\gamma}\Delta t \end{pmatrix} \tag{2.24}$$

Using the procedure above for each particle we can generate samples from the belief $bel(x_{t-1})$ according to the motion model.

2.5 Mapping

2.6 Mapping

The problem of robotic mapping is that of building an environment representation suitable for the tasks the robot has to carry out. The robot uses its sensor to perceive the outside world, but as we said in the introduction sensors are affected by noise. A good mapping algorithm should therefore cope with noise, while at the same time extracting relevant information from the sensors data stream. Another difficulty in the mapping problem arises from the high dimensionality of the maps space. For example, trying to build a build a very small map represented as 3×3 grid of binary cells involves search in a space with nine dimensions and 2^9 admissible configurations. Other issues are i) the correspondence problem, in which the robot should try to determine if sensor data taken in different points in time refers to the same object, and ii) the environment dynamics, where even small world modifications may invalidate a previously built map. In this section we will give a brief overview of the mapping problem, focusing on an approach we will use in the following, and we refer to [18] for details and further information.

One of the most widely used environment representation is the *occupancy grid* map [19]. According to this representation, the world is modelled as a grid made by binary cells[2]. Each cell $m_{i,j}$ may be either free or occupied, thus we need to estimate the probability $p(m_{i,j} = occ|x_{1:t}, z_{1:t})$. This is solved using a variant of the Bayes filter algorithm, namely the *binary Bayes filter*. It is often written using *log odds*, i.e. $\log \frac{p(x)}{1-p(x)}$, to avoid numerical instabilities due to near-zero probabilities. Using a procedure similar to the one in eq. 2.7 we get:

$$\begin{aligned} \log \frac{p(m_{i,j}|x_{1:t}, z_{1:t})}{1 - p(m_{i,j}|x_{1:t}, z_{1:t})} &= \\ &= \log \frac{p(m_{i,j}|x_t, z_t)}{1 - p(m_{i,j}|x_t, z_t)} + \log \frac{1 - p(m_{i,j})}{p(m_{i,j})} \\ &+ \log \frac{p(m_{i,j}|x_{1:t-1}, z_{1:t-1})}{1 - p(m_{i,j}|x_{1:t-1}, z_{1:t-1})} \end{aligned} \tag{2.25}$$

that is a recursive formula. By using the prior $p(m_{i,j}) = 0.5$, the middle term in equation 2.25 disappears.

The main problem related with the occupancy grids approach is that it needs the poses $x_{1:t}$ to be correct. Therefore we need to integrate the above mapping algorithm in a localization algorithm. A simple yet effective way to perform this is to insert the occupancy grid mapping in the particle filter algorithm described in section 2.2, just after the weight calculation (line 7). Another approach, more

[2]Although 3D grids have been used [20], we will use only 2D maps.

computationally involved but more robust to the data association problem, involves the use of the EM algorithm [21] Here the goal is to iteratively estimate a map given the previous map estimation and the robot belief:

$$m^{[i+1]} = \arg\max_{m} E_{x_{1:t}}[\log p(x_{1:t}, z_{1:t}, u_{1:t}|m)|m^{[i]}, x_{1:t}] \tag{2.26}$$

The E step is performed by using the Bayes filter algorithm, while the maximization step is performed by considering each cell independent from the others (details and derivation may be found in [22]). The main drawback of this approach is that the mapping is performed offline, usually using one or more logs. Furthermore EM does not retain a full notion of uncertainty, but it uses hill climbing in the space of all the maps, stopping when it finds a local maximum. However this procedure has achieved noteworthy results even when closing loops in large environments, as we will show in the next chapter.

Part II

An Indoor Robotic Museum Guide Architecture

Chapter 3

Cicerobot

3.1 Introduction

Cicerobot (figure 3.1) is a robotic museum guide that operates at the Agrigento Archaeological Museum. Its environment is the large "Sala Giove", one of the most important rooms in the museum. Such room is divided in two floors, the second one being horseshoe shaped. Showcases are present near three of the room walls, while in front of the forth one there is a large statue named "Telamone".

The robot operates in the second floor, about $20m \times 18m$ large. It is an ActivMedia PeopleBot© equipped with a laser range finder, a sonar array, a color stereo camera and a touchscreen. The primary obstacle detection sensor, i.e. the laser, uses light for measuring range and so it cannot detect the showcases neither the glass panels that divide the second floor from the first one (see figure 3.8a). This means that the robot may run into one of the invisible obstacles damaging them and itself. Furthermore the are two staircases, still invisible to the sensors, which the robot must avoid.

The robot task is to guide visitors around the room explaining them the various exhibits. It has a database of items of interest, along with the related description. The user may interact with the robot using its interface to get information about the museum or some particular item. When a visitor asks to see an item the robot starts moving toward it, stopping when reaches the goal position and starting illustrating the item. The main source of interaction are perhaps the pre-defined tours: the robot has some plans that comprise a set of interesting items, each plan corresponding to a particular topic.

As described in chapter 2, the various sources of uncertainty are addressed by the use of probabilistic algorithms. Each time the robot queries its sensors to update the beliefs about its position in a Markovian fashion. In order to get an estimate of its position, the robot has to build a map of the environment. In this

Figure 3.1: Cicerobot while interacting with visitors

work we decided to use the metric representation of occupancy grid maps. The map is built off-line, using a set of training data previously collected. Planning and localization processes both use the map to solve their tasks.

Several public demos validated the capabilities of the robot to deal with issues related to perception, planning and human-robot interaction typical of museum tour applications. A first session of experiments has been carried out from March to July 2006. The second session, based on the architecture described in this chapter, started in November 2007 and ended in January 2008.

3.2 System Overview

architecture A diagram of the main system components is showed in figure 3.2. The main robot task is activated by user interaction. A tour comprises a set of exhibits to be visited. Each exhibit is associated with a point in the map where the robot should stop and describe it. Starting from the current robot position the planner generates a path that will pass through each point, and the controller is activated in order to move the robot towards each planned waypoint.

The localizer uses Monte-Carlo localization to track the robot position. This subsystem is identical to the one described in [17], where range data are gathered from the laser sensor. The motion model parameters are estimated using the same training data used to build the map. As the Monte-Carlo methods produce a multi-modal pose distribution, we need to extract a single pose in order to proceed with motion planning and control. Various methods exist, for example Gaussian extraction, density trees or kernel density extraction. We observed that the particles, once the filter converges to a solution, are distributed in a very

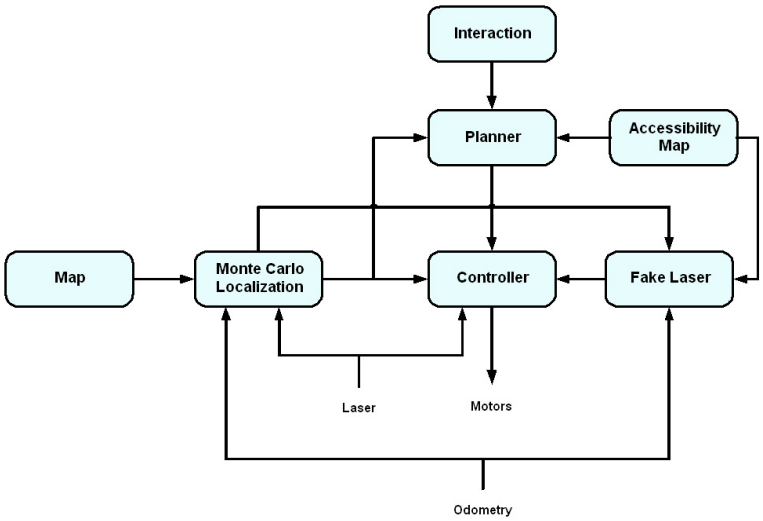


Figure 3.2: A diagram of the proposed architecture

narrow subspace of the poses, so the robot can just pick the particle with the greatest weight as a most-of-the-time correct pose estimation.

Planning and motion are achieved using deterministic algorithms, as they proved to be reliable for our task. Planning is performed after a set of goal points arrive, and it is not modified during the task execution. Although using a static plan may seem a limitation, the coarse resolution of the plan make it robust to dynamic changes in the environment. The only exception is when a goal point is occupied. In that case the robot will plan to a backup point (still defined in advance) and, if such point is occupied too, the current goal is skipped and the robot proceeds to the next exhibit.

The controller has the role of moving the robot towards a goal point. It uses information from the laser readings to avoid obstacles, and information from the localizer to adjust its heading towards the goal position. Furthermore robot dynamics are taken into consideration in order to generate motion commands not in conflict with physical constraints. Although the robot could be considered holonomic at low speeds, using dynamic constraints produces a smoother motion.

Once the robot has reached an exhibit, it stops and start describing it. At the time of writing we found no robust way to recognize item of interests, so the robot stops at fixed angles toward them. During traveling from one exhibit to the next one, especially during long paths, the robot generate random speeches ranging from casual comments to general information about the museum. This way we simulate a sort of interaction with the people, while at the same time keeping them from interfering with the robot motion (as it often happen, especially when

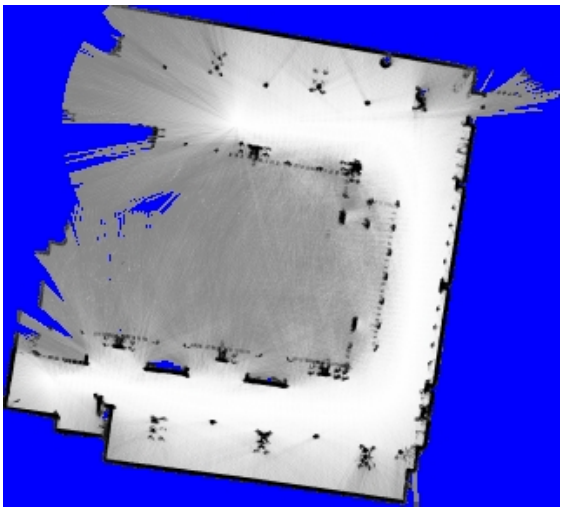


Figure 3.3: The "sala Giove" map. The whiter a pixel, the more probable it is of being free. Blue pixels are unexplored ones.

dealing with children).

3.3 Map building

Map building is performed using the EM algorithm outlined in section 2.6. We collected a data set of odometric measures and laser scans, and used it to build offline a map. The result is shown in figure 3.3.

3.4 Planning

Planning is based on the popular Wavefront planner. This algorithm uses an occupancy grid map bynarized so that each cell is either free or occupied. It accepts as input a goal point and output a map representation suitable for navigation. The main steps are illustrated in algorithm 2. Here lines $1-3$ initialize the variables, the set V keep tracks of all the visited cells, the set S is the current expanding front while the set D keep tracks of the distance of a cell from the goal. Lines $5-12$ update each cell near the cells in S; the function neighborhood(c, m) returns from the map m the set of the 4 cell near c, excluding the occupied and the out-of-bounds ones. The structure D may be easily used for navigation purposes, as the robot needs only to go from its current cell to the next lower one. Furthermore it needs to be computed only once per map, so that the planner needs to compute only the path from the robot current position to the goal position. The set of all

Algorithm 2 WaveFrontPlanner(m, g)

```
1: V = g
2: D(i) = {0}∀i
3: S = {s_i | s_i ∈ neighborhood(g, m)}
4: while V ≠ m do
5:   for all s_i ∈ S do
6:     S ← S \ {s_i}
7:     for all n_j | n_j ∈ neighborhood(s_i, m) ∧ n_j ∉ V do
8:       V ← V ∪ {n_j}
9:       S ← S ∪ {n_j}
10:      D(n_j) ← D(s_i) + 1
11:    end for
12:  end for
13: end while
14: Return D
```

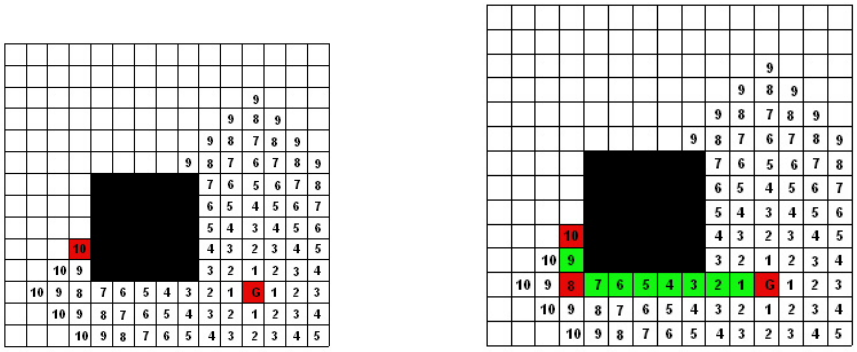
the cells in the robot path are called waypoints. The visual effect of the algorithm running is like a wavefront (hence its name) expanding from the goal point in all directions until the map is full covered (see figure 3.4).

We performed some slightly modification to the presented algorithm in order to cope with the museum environment. The first one is to grow all the obstacles by a factor[1], in order to avoid the robot passing too near to them. Although the controller is free to move near obstacles if, for example, it needs to avoid people, this solution helps the robot remaining in a safe area.

The second modification to the Wavefront algorithm concerns producing smooth movements. Using the naive implementation produces a path which is very discontinuous, often degenerating in staircase motion. Instead of keeping a number of waypoints equal to the number of cells in the path, we collapse all the waypoints along a line into two ones, indicating the two ends of the line. Thus starting from the robot position, all the waypoints which are "visible" (using ray casting) from the starting one are deleted, until the last visible one, which will be the starting point for the next collapsing procedure, until the goal is met. This procedure is illustrated in figure 3.4.and 3.5.

The output of the planner subsystem is a set of waypoints connecting the robot starting position to the next goal. The same procedure is repeated for each exhibit in the tour, obtaining this way the whole plan. We should note that, given our environment, the waypoints result very spaced. This produces a coarse plan that does not need to be modified if some occlusions (due mainly by people) happen

[1]This procedure is similar to the construction of the Configurations space for the robot.



(a) Expanding the wavefront. (b) The planned path.

Figure 3.4: The wavefront planner applied to a fictitious scenario. The red square with a G inside is the goal position, while the other red square is the robot position. Figure (a) shows the wavefront expansion, stopped when it meets the robot. Figure (b) shows the planned path, with a waypoint in the middle.

along the route: the controller task is to get rid of occlusions by moving around them, if possible, while moving the robot from one waypoint to the next one. An example of the plan generated in the museum is shown in figure 3.3.

3.5 Control

The controller task is to move the robot from one waypoint to the next one, while avoiding obstacles along its route. The approach we will describe has been introduced in [23], and has undergone some modifications in the next years. We will use the naive approach, with some slight adaptations to fit our application. The key concept is that robot movement may be controlled by issuing direct rotational and traslational velocity commands, thus controlling the two variables v, ω. If both velocities are kept constant for an amount of time, the robot will follow a circular trajectory whose curvature is given by v/ω (positive curvature means clockwise motion). Given the current goal position and the local obstacle configurations, the controller seeks to optimize a function over the (v, ω) space, i.e. generate curvatures that will bring the robot towards the goal while avoiding the obstacles. The function to be optimized is generally non-linear, so iterative methods may be applied. The drawback is that such methods may converge to a local minimum, thus obtaining sub-optimal solutions. Another solution arises when observing that the space of admissible velocities, given the robot physical constraints and its current velocity, is a small subset of the whole (v, ω) space. This insight is the basis for the development of the dynamic window approach.

Figure 3.5: An example of planned path in a real environment

3.5.1 The dynamic window

The dynamic window is a subspace of the robot velocities where the search for a function maximum may be carried. This space is obtained by combining three limitations given by the robot physical constraints. The first and most obvious one is that the motors speeds are bounded by a maximum speed:

$$V_s = \{(v, \omega) | 0 \leq v \leq v_{max}, -\omega_{min} \leq w \leq \omega_{max}\} \tag{3.1}$$

The second constraint is obtained by avoiding velocities that will bring the robot in contact with nearby obstacles:

$$V_a = \{(v, \omega) | v \leq \sqrt{2 \cdot dist(v, \omega)} \wedge |\omega| \leq \sqrt{2 \cdot dist(v, \omega)}\} \tag{3.2}$$

where the function $dist(v, \omega)$ returns the distance between the curvature given by v and ω and the nearest obstacle. The last and more important constraint is given by the robot acceleration. If in a given time t the robot velocities are (v_t, ω_t), then applying the maximum acceleration $(\dot{v}, \dot{\omega})$ for a time interval Δt results in the following constraint:

$$V_d = \{(v, \omega) | v_t - \dot{v}\Delta t \leq v \leq v_t + \dot{v}\Delta t \wedge \omega_t - \dot{\omega}\Delta t \leq \omega \leq \omega_t + \dot{\omega}\Delta t\} \tag{3.3}$$

Stated in other therms, if the controller acts every Δt time intervals, then it can apply only velocities contained in V_d.

The final search space is given by the intersection of the above sets:

$$V_r = V_s \cap V_a \cap V_d \tag{3.4}$$

As it can be seen it is much more smaller than the original search space, easing function optimization a lot.

3.5.2 The goal function

The controller task is to find a value in V_r that maximizes a goal function. Such function is composed by three distinct subgoals, namely *speed dist* and *clearance*, all normalized to 1:

$$G(v,\omega) = \sigma(\alpha \cdot speed(v,\omega) + \beta \cdot heading(v,\omega) + \gamma \cdot dist(v,\omega)) \tag{3.5}$$

where α β and γ are three coefficients which sum to 1 and σ is a smoothing function.

Speed is simply the projection of the v component over the (v,ω) space:

$$speed(v,\omega) = \frac{v}{v_{max}} \tag{3.6}$$

Heading depends on the curvature and the target heading θ_g. The role of this function is to generate rotational velocities that will bring the robot oriented toward the goal in a time interval[2] Δt. An exponential factor is introduced which forces the robot to rotate quickly towards the goal if it is very misaligned, but flattens the region around θ_g, to let the robot be free to move around obstacles:

$$\begin{aligned} e &= 1 + \delta \left(\frac{|\theta_g|}{\pi} \right)^{0.5} \\ heading(v,\omega) &= e \left(1 - \frac{|\theta_g - \omega \Delta t|}{\pi} \right) \end{aligned} \tag{3.7}$$

In order to compute the distance between a curvature and an obstacle we approximate the robot shape as circular with radius r. Although this may be a simplistic solution, the only drawback is that the robot will further keep away from obstacles, a feature sometimes appreciated. An obstacles field is a set of circles centered in each obstacle point o_i, as detected by the laser rangefinder, with radius r. Then the function *dist* is the minimum distance between the curvature c and each obstacle o_i:

$$dist(v,w) = \min_i \frac{dist_c(c,o_i)}{L} \tag{3.8}$$

[2]Here we assume that the robot will move with constant speed in the time interval Δt, which is a good approximation if we consider small time intervals.

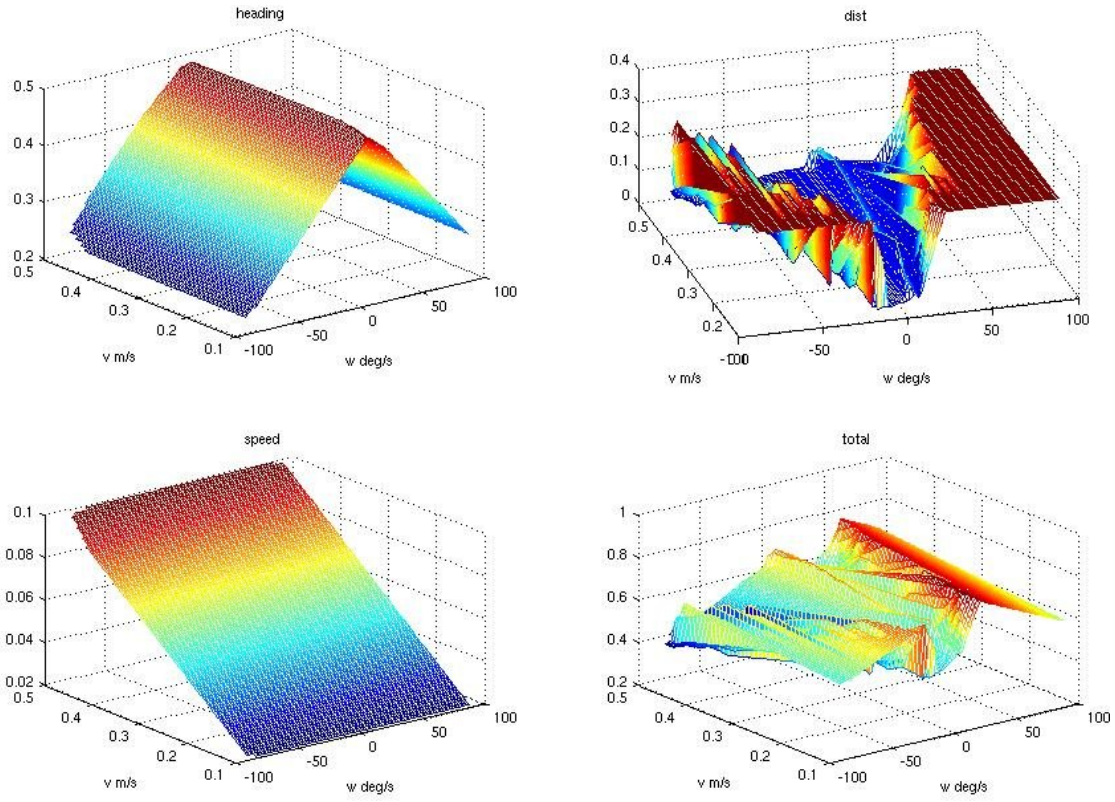


Figure 3.6: The goal function.

where L is the laser maximum range. In order to compute $dist_c$ we need to compute the distance between the circle defined by the curvature c and the obstacle circle o_i. This is performed by simple geometric computation.

Figure 3.7 shows the robot together with the detected obstacles, enlarged by the robot radius. In figure 3.6 the goal function together with the simple components is shown; here the target heading is $0deg$. In [23] the function maximum is found by exhaustive search over the discretized space. Here we used the Monte Carlo approach by generating random pairs (v, ω) over V_r for Δt time, then picking the best one. We found that, for standard computers, the results are even better than the standard approach given a small Δt.

3.6 Finding peoples

In chapter 2.3 we defined the sensor model as a combination of four probability distributions. One of them refers to unexpected short range readings. People surrounding the robot may cause such short readings, so a probabilistic filter that

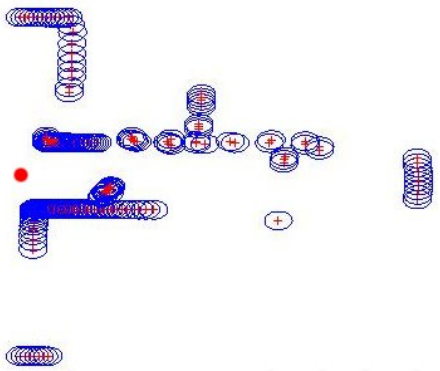


Figure 3.7: The robot (red circle) together with the obstacle field.

inspects the cause of a sensor data could detect them. In order to detect short readings we need to introduce a correspondence variable $\overline{c}_t^k \in \{hit, short, max, rand\}$. By applying Bayes rule and dropping irrelevant conditioning variables we obtain:

$$\begin{aligned} & p(\overline{c}_t^k = short | z_t^k, z_{1:t-1}, u_{1:t}, m) = \\ & = \frac{p(z_t^k | \overline{c}_t^k = short, z_{1:t-1}, u_{1:t}, m) p(\overline{c}_t^k = short)}{\sum_c p(z_t^k | \overline{c}_t^k = c, z_{1:t-1}, u_{1:t}, m) p(\overline{c}_t^k = c)} \end{aligned} \tag{3.9}$$

The prior distributions take the value $\{z_{hit}, z_{short}, z_{max}, z_{rand}\}$, so we need to solve only the first term in the numerator by integrating away x_t:

$$\begin{aligned} & p(z_t^k | \overline{c}_t^k = short, z_{1:t-1}, u_{1:t}, m) = \\ & = \int p(z_t^k | x_t, \overline{c}_t^k = short, z_{1:t-1}, u_{1:t}, m) p(x_t | \overline{c}_t^k = short, z_{1:t-1}, u_{1:t}, m) dx_t = \\ & = \int p(z_t^k | x_t, \overline{c}_t^k = short, m) p(x_t | z_{1:t-1}, u_{1:t}, m) dx_t = \\ & = \int p(z_t^k | x_t, \overline{c}_t^k = short, m) \overline{bel}(x_t) dx_t \end{aligned} \tag{3.10}$$

Using the probabilities $p_{hit}, p_{short}, p_{max}, p_{rand}$ we can devise the filter formula:

$$p(\overline{c}_t^k = short | z_t^k, z_{1:t-1}, u_{1:t}, m) = \frac{\int p_{short}(z_t^k, x_t, m) z_{short} \overline{bel}(x_t) dx_t}{\sum_c p_c(z_t^k | x_t, m) z_c \overline{bel}(x_t) dx_t} \tag{3.11}$$

Equation 3.11 does not possess closed-form solution, so we approximate it using Monte Carlo methods. This way, using the samples over the belief function, the integrals become a sum which is computable by a simple iteration over the particles. For each laser scan the probability $p(\overline{c}_t^k = short | z_t^k, z_{1:t-1}, u_{1:t}, m)$ may be thresholded in order to detect unknown obstacles, i.e. peoples.

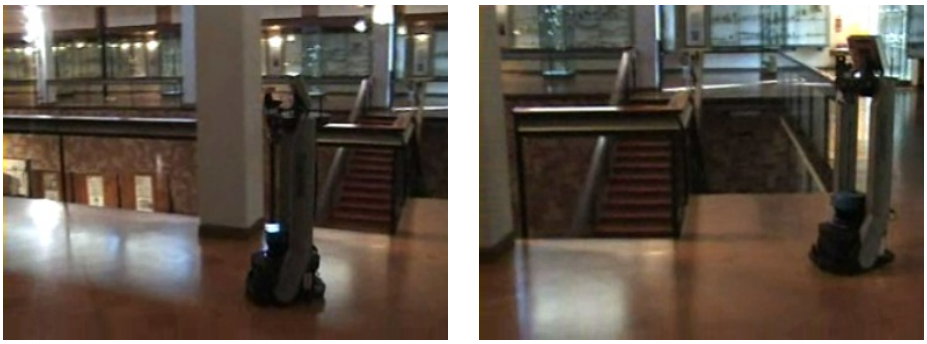


Figure 3.8: The robot near some dangerous places. Left figure shows the robot moving alongside a glass panel, which is invisible to the lasers. Right figure shows the robot passing near a staircase.

3.7 Dealing with invisible obstacles

As pointed out in section 3.1, the museum is full of showcases and glasses invisible to the laser light (figure 3.8a). Furthermore there are two staircases (figure 3.8b) which are not detectable by any range sensor, and which have to be considered as a serious danger for the robot.

In order to cope with invisible obstacles we created an accessibility map which masks every hazardous place. The original map and the new one share the same dimensions, so that a point in the first one corresponds to the same point in the second one. The robot keeps track of its position in either maps using the localization process. Each time a new position is computed by the localizer, the robot position in the accessibility map is updated accordingly. Fake laser readings are created by ray casting 181 beams in front of the robot pose. This way we simulate the robot moving and sensing in a fictitious environment where the hazards are masked with detectable obstacles.

The localizer process generates data at very low frequency compared to odometric sensor. As the dead reckoning system is accurate for small movements, we use odometric information to update the robot pose in the accessibility map until a new pose is generated by the localizer, in which case the stored robot position is replaced by the new one.

Finally we can combine the fake laser and the real one by taking the smallest of the readings corresponding to the same angle. The resulting hybrid laser is used as input for the dynamic window controller, which is able to avoid either fake obstacles (i.e. invisible hazards) either real ones.

As the whole system relies on the robot position for completing its tasks and also for safety, in case of catastrophic position errors the robot is halted and a

human operator must intervene to re-localize the robot.

3.8 Interaction with visitors

The robot is provided with a touch screen interface which is used to interact with the museum visitors. People may choose among a pre-defined set of tours, which comprise some exhibits grouped per thematic area. Each time the robot reaches an exhibit, it starts describing it. Usually it does not talk for more than 2 minutes to avoid boring people. A visitor may press a button to get more information, or another one to stop the robot talking. Because traveling from one exhibit to the next one may sometimes take time, the robot generate random speeches ranging from casual comments to general information about the museum.

If the robot is traveling and it detects some people, at random it may choose to stop and invite them in joining it. We found that this behavior is often amusing if not surprising, and it allowed the visitors to enjoy Cicerobot. On the other side the robot may ask the people to leave if they are blocking its way. This is performed in an incremental way: at first the robot talks in a gentle way, but if the path is still blocked then the robot will talk using an angry voice. We found that this behavior too is often amusing, while being useful for the robot in order to get a clear path.

3.9 Experimental results and Conclusions

Cicerobot was tested at the Archaeological Museum of Agrigento in a large room named "Sala Giove". It had run continuously for one week during the museum opening hours, with some pauses needed to recharge the batteries. It performed 37 tours in total, traversing a total of about 1.5km while often surrounded by people. During one of the tour the localization module failed, mainly because of hardware communication problems, and the robot needed to be re-localized. During the remaining tours the robot was fully autonomous while carrying its tasks.

People were often impressed by the robot behavior, especially while it told fictitious stories about itself or some ancient exhibit in the museum. Although children loved to block the robot path and to make its controller halt, the interaction module proved to be useful to let the robot move even if surrounded. Cicerobot angry voice was kept jocose, in order to not upset the visitors while still obtaining the desired results.

Although this work is not new in literature, it has been the basis for the further development we will see in the following. In the next chapter we will introduce a more robust control architecture with the aim of increasing fault tolerance and

robustness under unpredictable conditions. Furthermore the described mapping and localization algorithms will be used in the outdoor environments described in the next part, and as the basis for the introduced topological mapping.

Chapter 4

High level control

4.1 Introduction and motivations

In the previous chapter we developed a robotic architecture for an indoor museum-tour guide. We found that there are situations where the proposed system may fail, and recovering may be hard. For example if the robot looses track of its position it could be hard to get localized again, especially when the environment is very large. Trying to design a robot high-level controller which should cope with any kind of problem is simply unfeasible, as the world dynamics and inherent uncertainty may undermine any fault tolerance effort.

In this chapter we propose a robotic decision system that should act when unpredicted problems arise. We assume that the robot is provided with lower level modules which provide functions as planning, localization and motor control. The decision system plays its role when anyone of the lower level modules fails, or when the robot should take decisions based on a sensor fusion approach. For example, a museum tour-guide robot is often surrounded by people that block its way and generate false sensor readings. Altough filtering approaches have been proposed that eliminate false readings [17], we found that the localization module could still fail in certain situations. A decision system should analyze information like laser readings, velocity, people density and so on in order to choose an action. Moreover any action should have an expected outcome, based on current robot state and the future predicted state.

As all the above information are affected by uncertainty, a Bayesian network is well-suited to infer the robot state using sensor data. Moreover we used an influence diagram to describe the robot decisions and their outcomes based on the estimated state. We will show that even a simple network may have a surprisingly behavior when the robot is faced with hard decisions, making the whole system more fault tolerant.

4.2 Related works

Bayesian networks and influence diagrams have been applied to robotics mainly to monitor the physical state of the platform and to take actions according to the diagnosis. In [24] an expert system based on LIMID influence diagrams [25] is proposed. It is used to monitor the battery status of the robot, and a set of decisions is provided in order to act if a problem arises.

Plain Bayesian networks are also used to develop robot behaviors. In [26] a novel programming methodology is proposed, made by a multi-stage design process of Bayesian networks which are subsequently sampled in order to generate motor commands. Similar approaches are in [27] and [28].

Partially observable Markov decision process (POMDP) are closely related to decision diagrams, but they use an infinite horizon. A successfull application of POMDP to robot navigation is the robot Xavier [4]. It used a discrete representation of the environment to plan robot movements in order to minimize the travelling time. POMPDP have been used also on Minerva [6], a tour-guide museum robot, and in [5], where a nursery robot is proposed. In [29] POMDP are replaced by hierarchical dynamic Bayesian networks, resulting in a increased computational speed.

This work departs from the above ones in that we consider the localization and planning problems solved by lower level modules, while we deploy a decision system to act when the lower levels fail.

4.3 Modelling the Bayesian network

One of the main sources of fault is the presence of people around the robot. They generate spurious laser readings, making the localization harder. For example, the robot moving in an open space but flanked by two ranks of people may think it is running in a corridor. Once the robot position is lost, it is very hard to recover from the fault, especially in large dynamic environments. Altough this problem has been partially solved [17], during our experiments we found that localization may fail even when filtering out spurious readings. Other problems may arise when dealing with "invisible" obstacles, or when the set of parameters governing the low-level controller behavior makes the robot get stuck in a local minima.

A Bayesian network is well suited to fuse sensor readings in order to investigate the source of problems. We divided the set of nodes in two kinds, namely *sensor* nodes and *internal state* nodes. Using a medical analogy, a sensor node may be related to a test, and an internal state node to a disease. Using this analogy, in order to gain insight for the presence of a given disease we conduct some tests and report their results. A robot may query its sensors to estimate the presence

of a particular condition, and it may use the result of this test in order to make decisions.

We used not only physical sensors, but also "virtual" ones, i.e. sensors which report the result of some kind of elaboration on the real sensors data. For example, laser data and reported velocities correspond to physical sensors, while the degree of localization is a virtual sensor based on elaborations from the localization module. Internal state variables measure the belief the robot has about some unobservable state of the world. As we are most concerned with fault tolerance, we focused on two main kinds of fault sources, namely the presence of people around the robot and the proximity of obstacles.

When the robot is surrounded by people, it often happens that the degree of localization drops as a consequence of spurious laser readings, and the number of filtered out laser readings increases. Furthermore the controller slows down the robot in order to avoid the obstacles, while trying to find a clear path. These effects are modulated by the number and the relative position of the people respect to the robot: a few people blocking its path may slow down the movement, but the localization module may still keep track the robot position correctly. However many people laying on the robot sides may get the localization in trouble, but the robot could have enough space to continue moving.

As described in section 3.7, one of main problems of the robot museum environment is the presence of "invisible" obstacles. The robot is able to avoid them unless it knows its position in the map, but if the localization module fails nothing could prevent the robot from getting into hazardous situations. Dangerous situations may happen even if the robot is near unexpected obstacles, like people or unmapped obstacles.

From the above considerations we modelled the Bayesian network using the following inner state variables:

- **People**, with states $(none, few, many)$. It reports an estimation of the number of people surrounding the robot. Few people means also that people are not blocking the robot path, while many people implies that they are blocking its movements.

- **Lost**, with states $(true, false)$. It measures how much the robot is confident about its current pose estimation.

- **Danger Proximity**, with states $(true, false)$. It is used to detect if the robot is too near to unmapped obstacles.

- **Danger**, with states $(true, false)$. It is a general measure of how much the robot is running into troubles.

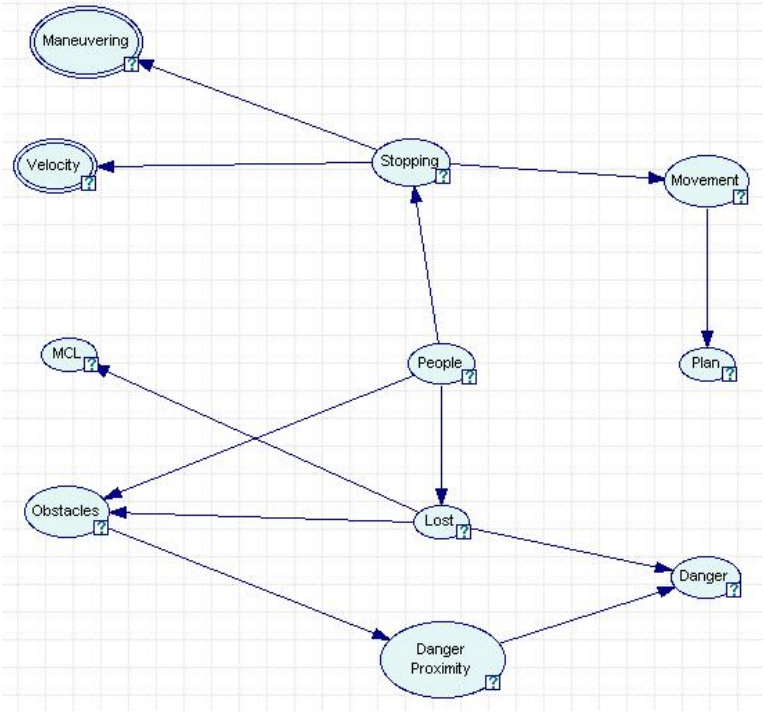


Figure 4.1: The Bayesian network.

People	*none*	*few*	*many*
true	0.2	0.4	0.8
false	0.8	0.6	0.2

Table 4.1: $P(Lost|People)$

People	*none*	*few*	*many*
Wanted	0.5	0.2	0.05
Unwanted	0	0.6	0.9
None	0.5	0.2	0.05

Table 4.2: $P(Stopping|People)$

- **Stopping**, with states $(none, unwanted, wanted)$. It reports if the robot is deliberately stopping or because of unexpected conditions (e.g. obstacles).

Moreover we need the following sensor variables:

- **Velocity** with states $(toolow, normal)$. It reports if the robot is travelling at normal speed or if it is stopping. It is a determinist variable, meaning that only one of the two states is allowed per time.
- **Maneuvering** with states $(true, false)$, which is true if the robot is conducting some kind of low-speed maneuvering. It is a deterministic variable.
- **MCL** with states $(localized, notlocalized)$. If the probability of reading a laser scan given the robot estimated position and the map is low, this variable shift towards *notlocalized*, otherwise it reports *localized*.
- **Obstacles** with states $(regular, unexpected)$. It is used like the MCL variable, this time reporting the output of the people filter.

The whole network is displayed in figure 4.1. Two examples of conditional probabilities distributions are given in table 4.1 and 4.2.

Many nodes propagate their state over time: for example, if in a given time t the robot detects people in front of it, in the next time step $t + 1$ it will be very probable that people will still be there. The same exists for the localization state. To model temporal interactions we modified the Bayesian network to include temporal arcs, making it a dynamic Bayesian network (DBN). The DBN unrolled for two time slices is shown in figure 4.2. We added two temporal links for *People* and *Lost*, while the other variables remained unchanged[1].

[1]The main reason for this is that all the other inner variables strongly depends on these two,

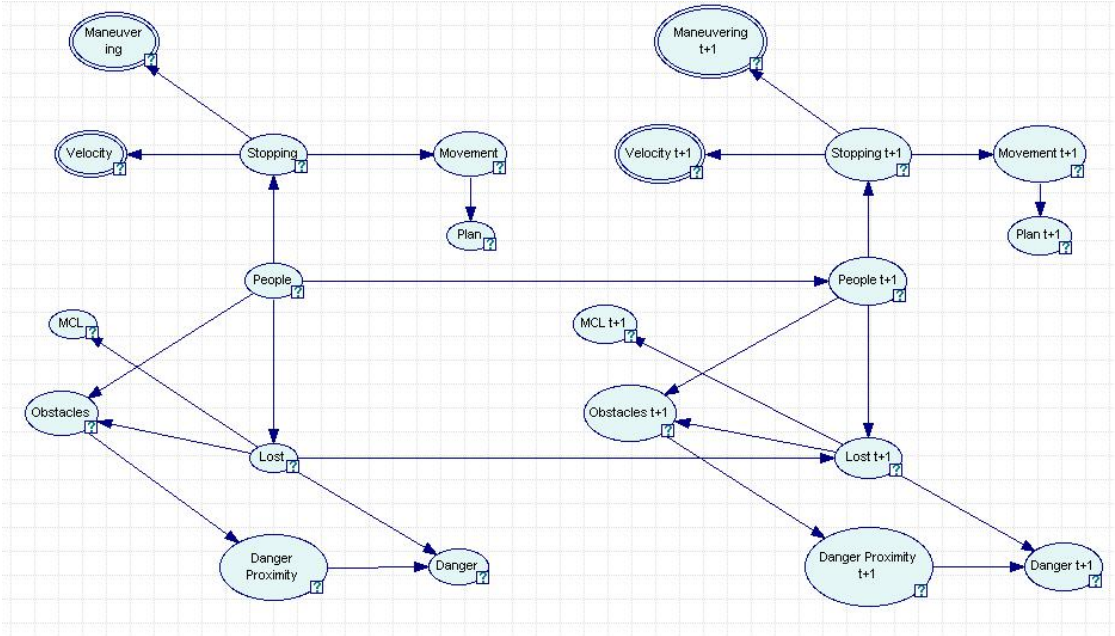


Figure 4.2: The dynamic Bayesian network.

4.4 Taking decisions

The robot should be able to make decision using the Bayesian network described above. To this aim we modified the network to include decision nodes, obtaining an influence diagram. Most of the decisions the robot has to face are deterministic, e.g. plan a path or stop to illustrate an exhibit, so we focused on decisions to recover from failures. As we said above we identified two main sources of failure, i.e. motion failure and danger failure.

Motion failure is most of the time due to people blocking the path. The robot may choose to try to avoid them, or simply ask them to clear the way. Each decision has its pros and cons: if the robot tries to maneuver around people the risk of getting lost could increase, but in many situations this is the most effective way to get rid of them. On the other side asking to leave may be the only option available, especially if many people are in front of the robot, while it does not always produce the desired result.

The resulting influence diagram is displayed in figure 4.3. We introduced a decision node, named *Decision*, with states $(moveaway, asktoleave, none)$, and three utility nodes. The first utility node, *Gain*, measures how much reward the robot gains by observing the people density before and after taking the decision.

so that no temporal link is strictly necessary.

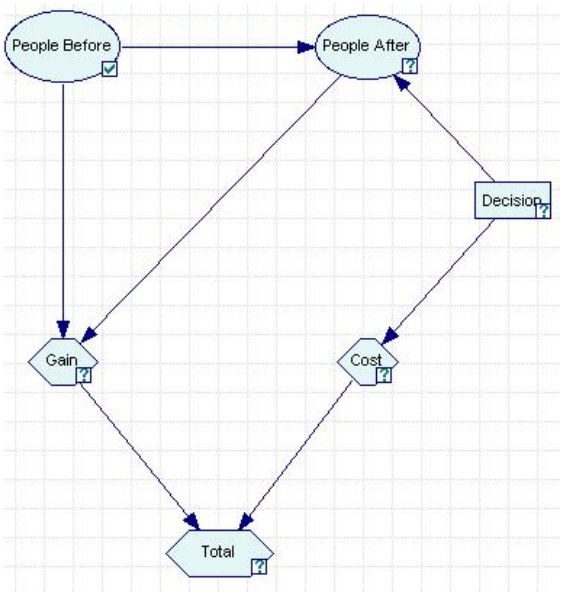


Figure 4.3: The influence diagram.

The second utility node, *Cost*, represents the cost of taking an action. Obviously outmaneuvering people is a task harder than asking to leave, while taking no actions has no cost. The last utility node, *Total*, is a simple linear combination of the twos.

Taking decision when the robot is in danger is a bit more complicate. The main source of danger is the loss of localization. At the time of writing we found no optimal policy to deal with this situation, so we decided to adopt a fixed strategy: at first, if there are people around the robot, it should ask them to leave. The next step is to generate random particles in the area the robot had last known positions recorded. Finally the robot starts moving at random until the particle filter converges to a correct pose. It usually needs moving only for a few meters to obtain a correct pose estimation. Although this may be a greedy strategy, we found it very useful when the localization module failed.

The final decision the robot has to take happens when a plan fails. This happens when the movement fails, i.e. when unexpected obstacles (i.e. people) are blocking the robot path. The first thing to do is to get rid of obstacles by using the policy described above. Then the robot uses its current position to obtain a new plan and continue its route.

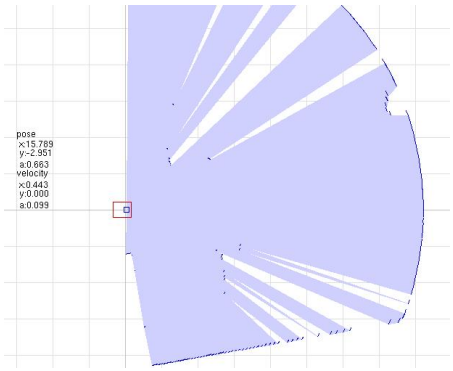
4.5 Experimental Results

The robot was tested at the Archaeological Museum of Agrigento in a large room named "Sala Giove". It had run continuously for one week during the museum opening hours, with some pauses needed to recharge the batteries. It performed 37 tours in total, traversing a total of about 1.5km while often surrounded by people. Here we show some snapshots of the decision system taken under different conditions.

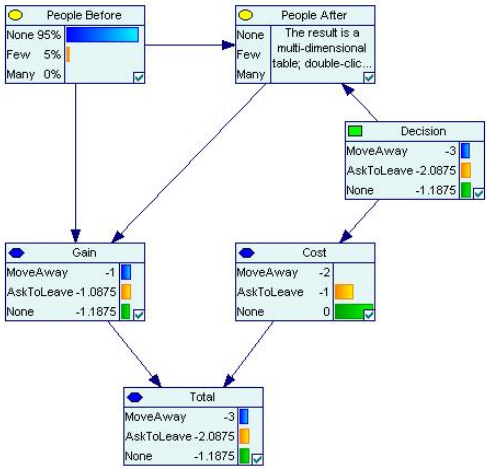
In the first snapshot (figure 4.4) the robot had clear path in front of it. The sensor system reported evidence that the robot was travelling at normal speed, no unexpected obstacles were detected and the localization was working fine. As we can see from figure 4.4c, the probability of people presence was near zero, so the decision system applied no recovery action (figure 4.4b).

In the second snapshot (figure 4.5) the robot path was slightly obstructed by people (marked with red circles). Evidence of normal speed (it was about $0.49m/s$) and good localization was reported in the Bayesian network, altough unexpected obstacles were detected. The probability distribution of the *people* node, given evidence, was $(0.37, 0.61, 0.2)$, correctly showing that few people were standing in the robot path. The decision system reported an utility of -0.085 if a *moveaway* action should be performed, against an utility of -0.1525 if *asktoleave* and an utility of -0.451 if *none* action should be selected. Obviously the best action was to avoid the people; under this decision the predicted probability distribution for the next time step people variable was $0.788, 0.157, 0.055$, thus a better situation was expected. Performing an avoid maneuver meant varying the low-level controller parameters to weight more the clearance function while weighting less the heading function (see section 3.5 for the meaning of these parameters).

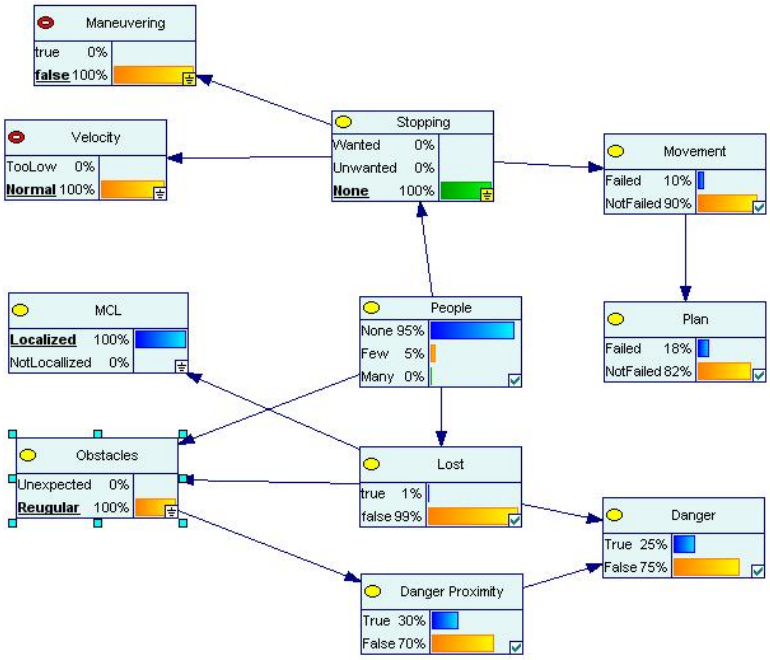
The last snapshot (figure 4.5) showed a situation where the robot was surrounded by many people, who were near enough to slow down the robot to a velocity of $0.177m/s$. This time we wanted to test the Bayesian network parameters, and we did not allow the sensor system to put evidence on the MCL and Obstacles node. The results were surprisingly good, as the localization and laser filter systems reported nearly the same results: the localization score dropped down to 30% while about 90% of the laser readings was filtered out (this is due to the poor localization score too). Under these conditions the decision system (figure 4.6b) chose to stop the robot and ask people to clear the way. In figure 4.6c it can be seen that the laser readings reported a large open area on the robot left. However, on the robot left side there was a screening of glass panels, invisible to the laser. If the controller had chosen to avoid people by maneuvering towards the open area, given the poor localization, the robot would have almost surely collided with the invisible obstacles. This kind of situation leaded us to the introduction of the proposed high level controller.



(a) The laser readings.

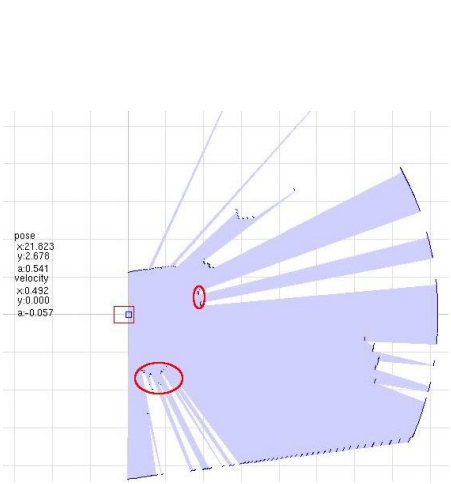


(b) The influence diagram.

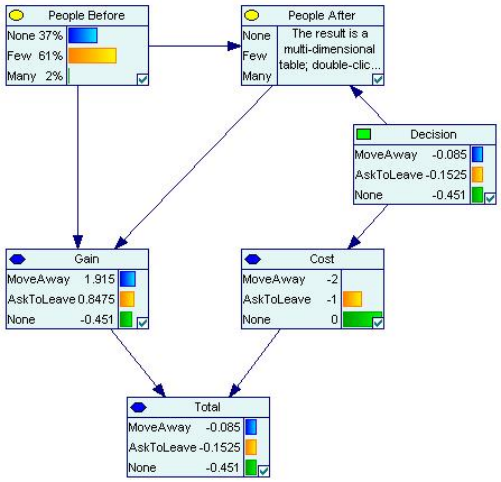


(c) The Bayesian Network.

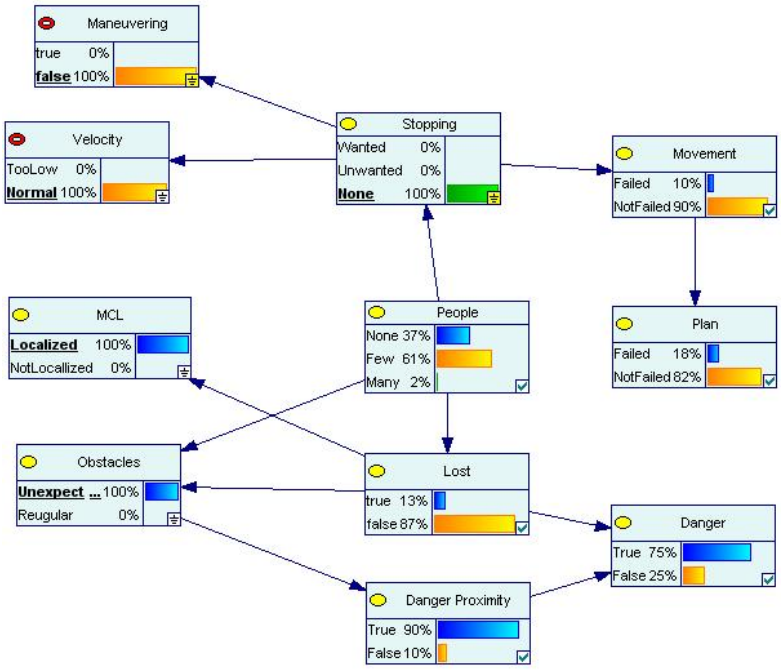
Figure 4.4: The robot has a clear path.



(a) The laser readings.

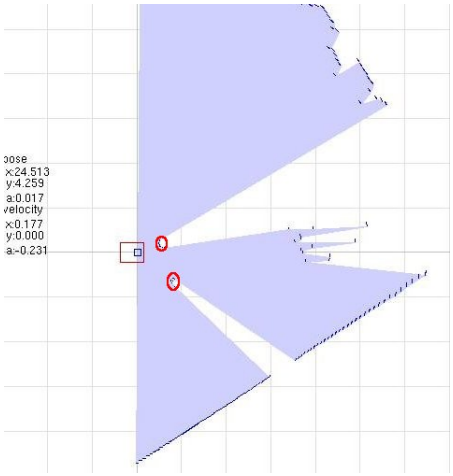


(b) The influence diagram.

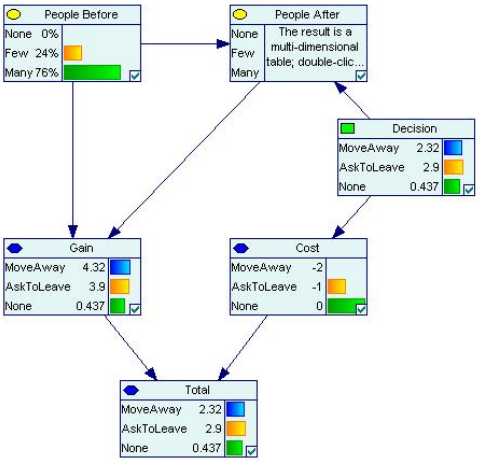


(c) The Bayesian Network.

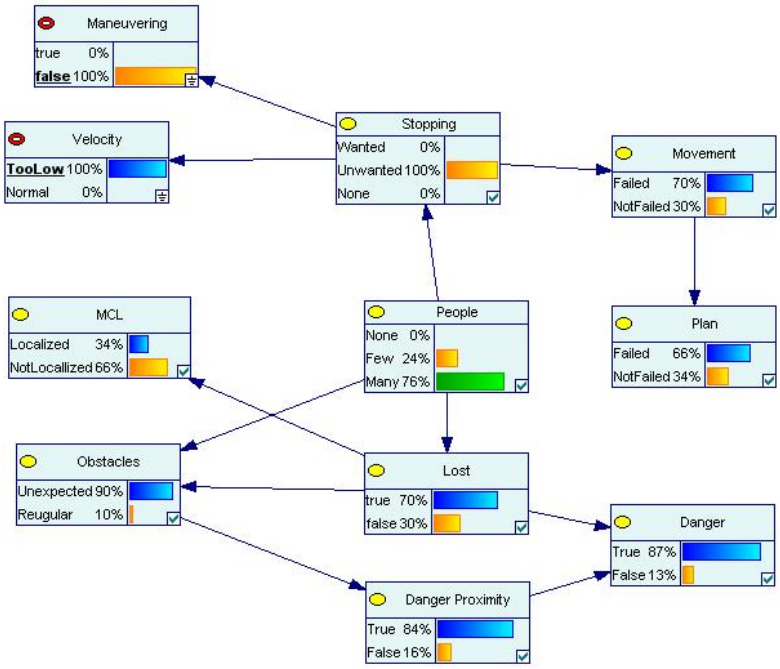
Figure 4.5: Few people block the robot path. The robot decided to move around them.



(a) The laser readings.



(b) The influence diagram.



(c) The Bayesian Network.

Figure 4.6: Many people block the robot path. The robot decided to stop and ask the people to leave.

4.6 Conclusions and future works

In this chapter we presented a high-level robot decision system based on dynamic Bayesian networks and influence diagrams. It has been applied on a tour-guide robot operating at the Archaeological Museum of Agrigento. The proposed system acts on top of a set of lower level subsystems (like the planner, the localizer and the controller) to monitor the robot state and act in case of troubles. We identified two main kind of faults, namely the inability to move because of obstacles and a wrong pose estimation. A dynamic Bayesian network monitors the robot state inferring possible causes of trouble from sensor readings, while a decision system based on an influence diagram act accordingly in case of troubles.

Experimental results showed that the fault tolerance of the robotic system improved, as the robot is able to decide the best action using policy evaluation and utility functions. We are currently working on introducing more variables in the network, in order to improve human-robot interaction. Furthermore we are investigating learning algorithms to automatically estimate the network parameters and the policy function. The ultimate goal could be a lifelong learning robot that adapts to the environment changes and users interactions.

Part III

Outdoor Robotic Museum Guide Architectures

Chapter 5

Basic Architecture

5.1 Introduction

In this chapter we present the first robotic architecture for outdoor environment: a Botanical Garden. Obviously we are focused on tours that are considerably simpler than those offered by human guides of the garden. Nevertheless delivering even simple robot guided tours is a challenging task due to the large spaces involved and the high noise in sensor readings. We made use of standard SLAM techniques, but we will point out where they fail and the proposed solutions.

The robot has two main goals, namely i) guide tourists along the botanical garden and ii) ensure the safety of people, environment and itself. During a route the robot will stop near exhibits of interest and speak about them. A planner generates the set of actions needed to accomplish its task (including the speaking actions). The controller executes the actions, while ensuring the safety of people, environment and robot, and monitoring that no problem occurs. Finally a localization system keeps track of the robot position, using a large scale topological map and different small scale metric maps.

The robotic platform we are working with is an ActiveMedia© Pioneer 3-AT (figure 5.2) using differential drive and equipped with a laser scan range finder, a sonar array, a stereo camera, a digital compass and a GPS. The environment map is shown in figure 5.1, the red thick lines displaying the robot route. The itinerary is composed in part by a pathway bounded by short walls. The environment is nearly $10000m^2$ large, although only a small part of it is traversable by the robot. Furthermore roads are often covered by foliage or mud. A typical route takes $\backsim 30min$ to complete it (not including speaking). The environment is highly dynamic as visitors are all day walking inside the botanical garden, gardeners work alongside the plants and even workers trucks happen to pass by. Any sensing and controlling process must therefore cope with such dynamics.

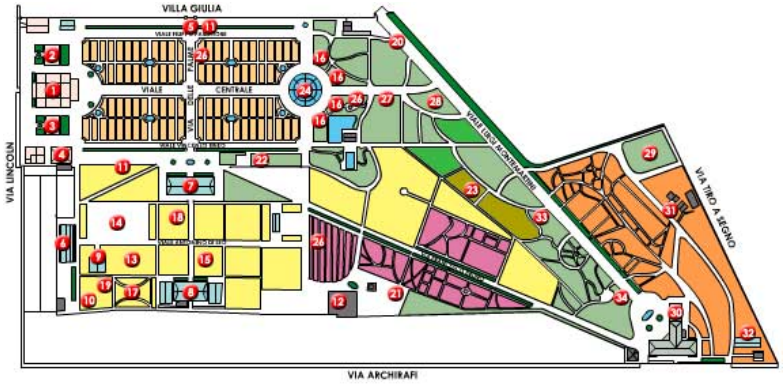


Figure 5.1: Part of the Botanical Garden Map. The thick red line displays the robot route.

Figure 5.2: A close-up of the robotic platform used.

5.2 Motivation and related work

Two early works about robotic tourist guide are in [6] and in [2]. They used probabilistic approaches to solve the localization problem. The pose of the robot is represented as a distribution over the space of configuration, and it is continuously updated using information gained from sensing and control using Bayesian filtering. Although noteworthy results have been reported in the context of localization in dynamic environments, problems arise when i) the robot motion model is very broad distribution compared to the sensors model, mainly because of a poor odometric platform, and ii) when closing loops in large scale environments [30].

An approach that tries to address both problems concerns splitting the environment map into several smaller maps and creating a graph structure to link them. This leads to hybrid metric-topological maps [31, 32, 33, 34]. Splitting the environment requires each small map being well-defined, and the union of all the maps being (not considering overlapping) the whole map. As we will point out in a while, this is a very hard requirement given the robot and the environment described.

A slightly different approach uses landmarks instead of metrics map, estimating the pose distribution using particle filters and Extended Kalman Filters (see for example [35]). The main issue related to this approach is the landmark detection and recognition. This is a problem easy to solve using a laser range finder[1] but very hard using computer vision in outdoor environment. Although visual slam is a topic in rapid development (see for example [36, 37, 38]), we did not find a suitable methodology able to cope with outdoor environment often cluttered with occlusions.

A completely different approach is in [39], where a cognitive architecture based on artificial consciousness is employed. Cicerobot used an internal simulator to plan and anticipate actions, while sensors feedback is used to correct its behavior. However from a practical point of view creating an accurate enough simulator for large scale environments is still an open issue.

Figure 5.3 shows raw odometric and laser readings, where large odometric errors are evident. It can be seen that after the robot travelled only for $50m$ the heading error is $\backsim 70°$. The arc-shape of the pathway is due to a constant drift to the right during straight motion, of which the odometric system is not aware, while it being aware of the correction made to keep the robot in the pathway center. Trying to generate a complete map using state-of-the-art SLAM algorithms (see next sections) leaded to very poor results. The problem lies in the poor

[1]The range finder is commonly used to detect trees in the environment. In the botanical garden of figure 5.1 there are not such easy detectable features.

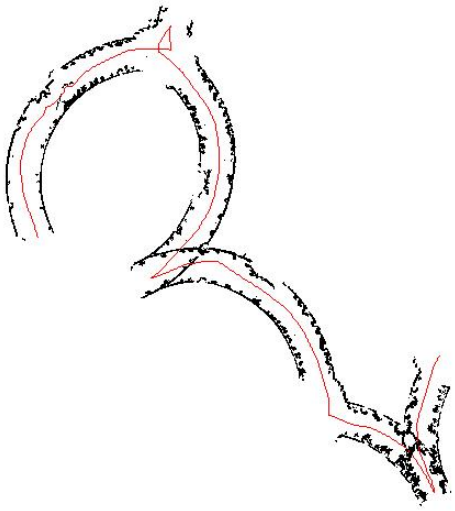


Figure 5.3: Raw odometry with laser scans. The two red circles point to the same position, i.e. the central plaza. The distance between the two points in this map is $\backsim 100m$.

.

odometry (although a motion model has been estimated using real data [40] to take into account systematic errors.) and in large laser noise inducted by the strange reflection of plants and trees. This leads to the practical impossibility of generating metric maps from raw data of much of the environment.

GPS data were not much more precise, as shown in figure 5.4. Satellite communication was often hindered by the intricate leafage of tall tree, leading to errors often as high as $10m$. Furthermore heading can not be estimated using only the GPS, and the digital compass is unusable while the robot is in motion (although it is very accurate if the robot is stopped).

Such results lead to the conclusion that integrating the robot pose continuously as it travels is infeasible, given the environment and the robotic platform described. The proposed approach relies therefore on a "lazy localization", i.e. the point-wise correct pose is computed only when useful and feasible. The usefullness of a correct pose is due to the robot task, i.e. it is needed (together with a correct metric map) only when the robot should speak about an object of interest. In the following setions we will describe the proposed system and the experimental results.

5.3 System Overview

The proposed system is composed by three main components, being i)the path planner, ii)the controller and iii)the localizer. Shared among them lies the envi-

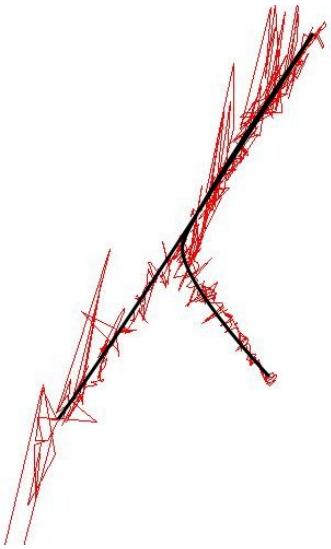


Figure 5.4: GPS data obtained during the same path as in figure 5.3. The black line displays the true robot path.

ronment map, represented as a directed graph. As the robot receives a route the path planner generates a plan made by a set of nodes to visit. The controller goal is to move the robot from one node to the following one, querying the localizer for the actual position and acting as soon as a problem arises (it happen very often due to localization problems and dynamic environment). The localizer task is to keep track of the robot position, either in the sparse discrete space of the graph either in the fine-grained low-level map if it is using it.

5.4 Environment representation

The environment map is represented as a directed graph where each node is a place meaningful for the robot. A place is meaningful if there is an exhibit, or there is a junction in the path. An edge exists between two nodes a and b if it possible to travel from a to b without passing by any other node[2]. Each edge contains the relative orientation between two nodes, taken using the robot digital compass.

A node represents a subset of the whole environment, although the set of all the nodes does not sum to the whole map, i.e. there are large portion of the environment which are not covered by a node (sometimes neither by an edge). This is due to the hardness of generating a map in some areas (for example the large plaza in the center of the pathway), or to the poor localization performances

[2]The resulting graph is doubly connected, however we will keep the directed formalism because each link carries different information.

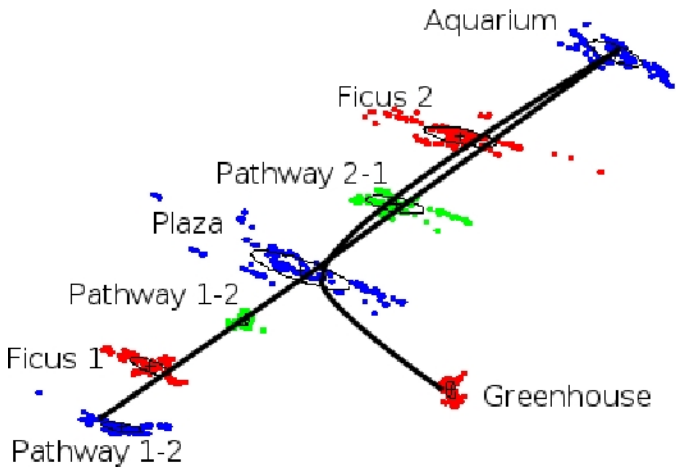


Figure 5.5: GPS data collected during map building. Each node has a name, and a gaussian distribution whose contour is displayed as a black ellipse. The black thick line is the $250m$ long path the robot followed while collecting the data.

if surrounded by too much people. During the map building process we collected GPS data for the whole route, stopping it for a while along meaningful places to collect more data. For each node i we fitted a gaussian distribution memorizing its mean μ_i and its covariance Σ_i. The results are showed in figure 5.5.

The places which contain exhibits the robot should talk about include a metric map, like the two Ficus in figure 5.7. Experimental results show that the robot was able to localize itself using these small maps, even if surrounded by people.

5.5 Path Planning

Path planning is performed by simple graph search over the environment map. Each time the robot receives a new route made by a list of exhibits, a list of nodes to visit is extracted. The planner then generate a set of high level actions a_i to perform, i.e. the list of actions needed to move from the node the robot belongs to the next ones.

When the robot reaches an exhibit node, the local metric map is used and a new path is computed this time using cells instead of nodes. Once the goal position is reached the robot starts illustrating the exhibit.

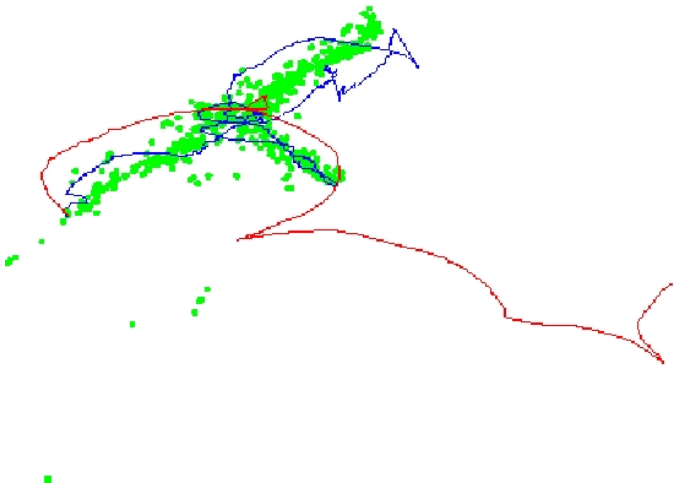


Figure 5.6: Comparison of GPS (green dots), raw odometry (red line) and GPS with EKF (blue line). Refer to figures 5.3, 5.4 and 5.5 for comparison.

5.6 The localization system

The localization task is performed using the topological map or the metric map if it is available. The GPS plays a key role during the large scale localization, as it is unaffected by occlusions and its error is bounded.

The robot keeps track of its pose using an Extended Kalman Filter, where the prediction step is performed using traslational and rotational velocity, and the measure integration step is performed using plain GPS data. Figure 5.6 shows data from odometry, plain GPS and the EKF. Results are still inaccurate, mainly because of the burst errors in the GPS, but they are far better than using plain GPS or plain odometry.

Let $N_t \in 1 \dots N_p$ be the current node at time t and $x_{1:t}$ the set of positions collected until time t and integrated using the EKF. Let furthermore $a_{1:t-1} \in 1 \dots N_p$ be the high level actions applied until time $t-1$. Each a_i represents the act of moving the robot from its current node towards the next one (this is very different from the low-level control u we will introduce below). The set of high-level controls is usually planned in advance during the route definition.

We can recursively compute the probability of being in a node n, given all the observations and control, by applying Bayes rule and dropping conditionally independent variables:

$$
\begin{aligned}
&P(N_t = n|x_{1:t}, a_{1:t-1}) \propto \\
&p(x_t|N_t = n, a_{1:t})P(N_t = n|x_{1:t-1}, a_{1:t}) \\
&= p(x_t|n)P(n|x_{1:t-1}, a_{1:t-1}) \\
&= p(x_t|n)\sum_{n'} K(n', n, a_{t-1})P(n'|x_{1:t-1}, a_{1:t-2})
\end{aligned} \tag{5.1}
$$

Here we assume that the pose is independent of the node given the control, and that past controls give no information about the node transition.

The variable $K(n', n, a) = P(N_t = n|N_{t-1} = n', a)$ is a transition matrix used to introduce stochasticity in the control. This means that the robot may decide to move from one node n' towards a node n by applying control a, but the probability of entering node $q \neq n$ is non-zero. The transition matrix has been built empirically for each node by observing the robot behavior during several runs. We will see the usefulness of such approach in the experimental results. The distribution $p(x_t|N_t = n)$ is the likelyhood of the pose given the node. As stated before this is a gaussian distribution with mean μ and covariance Σ, so that:

$$p(x_t|N_t = n) \sim \mathcal{N}(x_t; \mu_n, \Sigma_n) \tag{5.2}$$

Tests performed using this system showed an error not larger than $3m$, mainly due to the satellites communications. These errors were evident during the transition between two nodes, but the whole system is mainly unaffected by them. Sometimes it happened that the robot missed completely a node, or picked up the wrong one: this was the kind of problems the controller has to cope with.

If the node includes a metric map, the robot will use it to localize using a precision lower than $10cm$. The maps are created using Monte-Carlo occupancy grid mapping. We will briefly describe the problem and the proposed solution. Further details may be found in the huge literature on this topic.

The SLAM (simultaneous localization and mapping) problem involves estimating the posterior of the robot pose along with the map:

$$p(x_t, m|z_{1:t}, u_{1:t}) \tag{5.3}$$

where $x_t \equiv (x, y, \theta)^T$ is the robot pose, m is the map (usually an occupancy grid map), z are the measurement (in this case the laser readings) and u is the low-level control (in this case traslational and rotational velocity). Equation 5.3 may be factorized in:

$$p(x_t|z_{1:t}, u_{1:t}) \cdot p(m|z_{1:t}, x_{1:t}) \tag{5.4}$$

The second term in equation 5.4 may be easily computed using an occupancy grid mapping algorithm. The first term is computed using a particle filter that incrementally updates the pose x_t using measurements and control variables.

Particle filters need a proposal distribution to resample during each step. A common choice is $p(x_t|x_{t-1}, u_{t-1})$ i.e. the motion model. Although easy to compute, it is not well-suited for this kind of problem. Another kind of proposal is $p(z_t|m_{t-1}, x_t)$, which is harder to compute but gives much better results than the former. In [41] a system is proposed that performs well even in large scale environment, although the assumptions on the motion model are a way too restrictive for our robot.

As soon as the robot enter a node with a map, an initial belief is generated by sampling particles in the neighborhood of the entry point, and the particle filter is started. It takes usually less than $2m$ of travelling for the particle filter to converge to the true pose, and it is hard to have the filter diverged. If the localization is performing bad (a good indicator is the mean weight associated with each particle) random samples are generated in the map, so that the robot will recover its position even after completely loosing it [42].

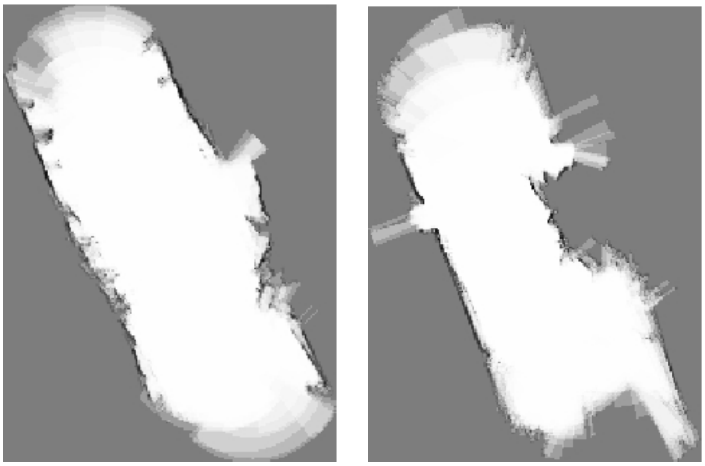
5.7 The controller

The controller goal is to move the robot from one node to the next one, or to move the robot towards a specific point in a grid map. We used the dynamic window approach described in [23]. Here a brief explanation of the approach is provided, and we refer to the original paper for details.

Robot movements are performed by issuing direct rotational and traslational velocities, thus controlling the two variables v, ω. If both velocities are kept constant for an amount of time, the robot will follow a circular trajectory whose curvature is given by v/ω (positive curvature means clockwise motion). Given the current goal heading and the local obstacle configurations, the controller seeks to optimize a function over the (v, ω) space, i.e. to generate curvatures that will bring the robot towards the goal while avoiding the obstacles. The space of admissible velocities, i.e. the space over which function optimization is performed, is a small subset of the whole (v, ω) space when considering the robot physical constraints and its current velocity, . The goal function is composed by three distinct subgoals, namely *speed dist* and *clearance*, all normalized to 1:

$$G(v, \omega) = \sigma(\alpha \cdot speed(v, \omega) + \beta \cdot heading(v, \omega) + \gamma \cdot dist(v, \omega)) \qquad (5.5)$$

where α β and γ are three coefficient which sum to 1 and σ is a smoothing function. *Speed* is simply the projection of the v component over the (v, ω) space. *Heading* measures the misalignment between the robot and the goal heading. During travelling between nodes the heading stored in the map edges is used, while during the navigation in the grid map the heading is computed using the robot position and the goal point position. Finally the function *dist* is the minimum distance



(a) Ficus 2 (b) Ficus 1

Figure 5.7: Two maps generated for two nodes.

beetween the curvature v/ω and each obstacle o_i as detected by the laser range finder.

The topological map has been built in a way that if two nodes i and j are linked, it should not happen that during the travelling from i to j a third node $k \neq i, j$ is detected. Howewer as pointed out in the previous section, it may happen that the localization reports a wrong node. If this happens the controller stops any travelling behavior, asks the planner for a new plan from the node k to the node j and restarts. In the worst case (a GPS burst error for example) this will lead to the robot starting-aligning-restarting a couple of times, without any other consequence for the localization and the controller system.

The metric map associated with the neighborhood of an object of interest is used to place the robot in an exact position and to turn it to a desired heading. This is useful to let the visitors understand what the robot is talking of. Once the pose of the robot is well known (the particle filter converged), a path is computed using the grid map. As soon as the robot reached the desired location (or the nearer one, if it is occupied), it will stop, turn towards the object of interest, and talk.

5.8 Experimental Results

The first part of this work concerned generating metric maps of the environment and testing the localization system. A large map comprising a pathway (about one third of the whole route) is displayed in figure 5.8. That map lacks many of the fe-

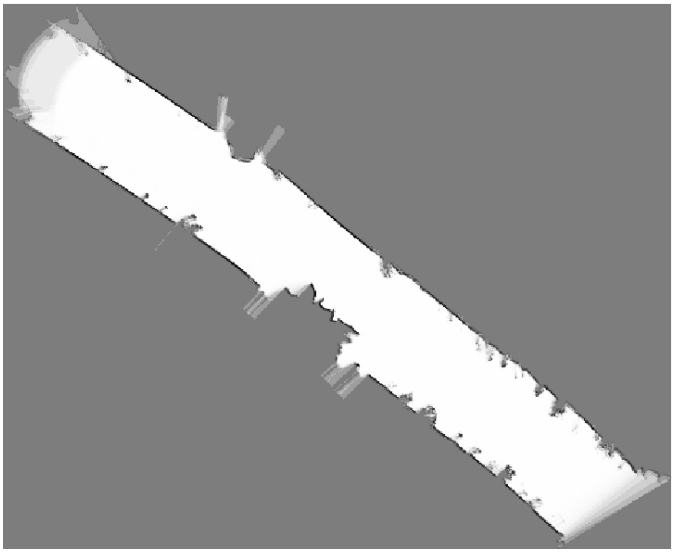


Figure 5.8: A large map displaying the first pathway. It lacks much details, including numerous plants in front of the two lateral walls.

Figure 5.9: Images taken during a crowded demo.

atures in the real environment, mainly the plants and small trees in the front of the side walls, as they were "overwritten" during the SLAM process. Large odometric errors and the presence of people leaded to very poor localization performances in the large map, sometimes degenerating in the robot being completely lost. This test showed that using a single large map of the environment was unfeasible, thus leading to the approach we proposed.

Although we managed to obtain a map of a pathway, obtaining an accurate map of the central plaza was impossible, mainly because of large spaces and poor odometry. A similar problem has been reported in [6], where the authors used a camera pointed towards the ceiling to aid the localization. Obviously we can not use the same approach in our environment.

Different results were obtained by reducing the map size. In figure 5.7 two maps corresponding to the neighborhood of two large trees (named "Ficus") are shown. The robot was able to localize itself using these maps even in highly dynamic environment. These maps were used as local metric maps for two nodes.

We performed a second set of tests in order to validate the high level localization system. At first we used no transition matrix, thus considering deterministic motion. The localization system performed well where no obstacles were on the robot path. Different results were obtained in dynamic environments. Here the robot tried to move around people, loosing track of its position very often. Furthermore the deterministic nature of the motion model made error recovery impossible. When using the transition matrix the uncertainty in motion leaded to a multi-hypothesis belief, thus enabling error recovery when new GPS data arrived.

The real testbed for the proposed system was during an international conference held in the Botanical Garden. The robot performed several tours, covering more than $3km$ in a day. Figure 5.9 shows some snapshots taken during the route. The robot was equipped with a text-to-speech system, tuned to produce a robotic voice.

An example of planned route is showed in figure 5.5. Apart from the pathway nodes, the robot had to stop and talk in all the other ones. Talking was performed also while travelling from one node to the next one.

Position errors happened in the node named 'Plaza'. Here the robot was surrounded by people, and it believed it was in the wrong node while trying to move around them. Subsequent path planning and re-localization processes corrected the robot pose, demonstrating the system high fault tolerance and robustness.

5.9 Conclusions and future works

In this chapter we proposed a robotic tour-guide for outdoor environments operating in a Botanical Garden. We can easily affirm that simply putting a robot in the context of a museum generates a new interest in visitors, even the ones

completely unaware of the role of the robot, as we often have experienced in the pauses between experiments.

The main problem related to this environment was high noise in both the proprioceptive and exteroceptive sensor data. We performed tests using state-of-the-art SLAM algorithms, noting their failures and how to cope with them. In particular we noted that integrating continuously the robot pose is often infeasible in our environment, and we proposed a "lazy localization" approach that proved to be reliable and fault tolerant, even in highly dynamic environments.

This project needs anyway to be enhanced under different aspects. As the planned future routes will comprise paths harder to follow in the Botanical Garden, we need to improve the controller and the localization system. In particular, it would be easier to detect an object of interest using vision and to move towards it, instead of using a metric map to track the position. This way the robot will be more "aware" of what it is talking of. Furthermore we are studying a way to detect a drivable surface using vision, thus avoiding potential invisible hazards. Finally we are studying the integration of a GIS (global information system) to aid the navigation and localization systems.

Chapter 6

Appearance Based Navigation

6.1 Introduction

So far we assumed that the process of map-building is splitted from the process of localization. Furthermore we used raw sensor data (laser and GPS) without any feature extraction or preprocessing. In this chapter we explore an emergent mapping methodology, namely *appearance based navigation*. Under this framework the robot recognizes places using camera images, while at the same time building a topological representation of the environment. Although the proposed system is not used yet in the context of robotic museum guides, it poses some basis for the chapter 7, in which places recognition and topological maps are the main topics. Furthermore the proposed algorithms are appliable either in indoor and outdoor environments, thus bridging the gap between different methodologies.

Our primary goal here is to build an incremental model of the environment as the robot starts without any prior knowledge. Robot sensing is assumed stochastic as it is often heavily corrupted by noise and dependent on environment conditions. Our system goal is to detect places which share a common representation and which are close in the space, using camera images and odometry information. The places are modelled in a holistic fashion [43], thus not relying on landmark or local image information. They are furthermore organized in a topological map, which forms the basis for a Markov model of transition from a place to another one. Although it is widely common to use a-priori information (such as a training set of the pattern the robot should recognize), we let our robot to discover and model its environment without such information. Therefore a great effort has been dedicated to incremental statistical learning, which starts from features detection to place shaping.

Although the literature in appearance based navigation is under rapid development, little efforts have been devoted to on-line learning of environment. The

usual approach is to learn a set of features taken from a previously filled database of sensor readings, then let the robot recognize them. Machine learning algorithms provide robust ways to achieve such task, but they lack incremental counterparts.

In our work we tried to address this problem by adapting batch algorithms (like EM) to on-line ones. We did not give the robot any prior knowledge about the world. Furthermore no human intervention (like labelling or setting the number of cluster) is required to run our system. This poses some non-trivial problems, related to dimensionality reduction and to the concurrent processes of learning and recognition. The use of on-line learning algorithm often leads to radical changes in the methodologies to be applied. Anyway we tried to adopt well-known although still experimental algorithms to keep our system robust and reliable under varying conditions.

The problem of finding new places is still an open one. Some earlier approaches used to place nodes at equidistant locations. We chose to merge odometric and visual information to achieve a more flexible node placing. Although odometry is known to be unreliable for position tracking, it is still useful during short-length run, as we will prove in the subsequent sections.

6.2 Related works

Appearance-based robot navigation has attracted a lot of attention because of its simplicity and success in localization problems. In [44] a comparison between some appearance-based and landmark-based navigation systems has been analyzed. It is shown that the former approach is more robust to noise and occlusions.

Omni-directional images have been widely used [45, 46], although they are more sensible to dynamic environment. In [47] this problem is addressed by introducing a biological inspired attention system in the context of novelty detection.

One of the major drawbacks in appearance-based robot navigation is the need to train the system off-line with images describing all the environment often under varying conditions. This is usually due to the need of input data dimensionality reduction, in most cases performed by principal component analysis (PCA). Although incremental versions of PCA exist (see [48] for example[1]), their application to pattern classification is still under development. Another successful is in [49], where appearance of places is modelled using a mixture model on Fourier coefficients.

A batch training process has been used with considerable success for instance in [50] and [51]. In the first one the system is trained with features vectors built

[1]Although the incremental PCA is shown to work as well as the batch one, the eigenspace changes at every update and so the projected coefficients, making them unusable for further training algorithms.

from a large database of images taken either in indoor and outdoor environment. The training is performed by PCA and Parzen window density estimation, while recognition is aided by a hidden Markov model for transition between places. In [51] features vectors are extracted using an underlying biological-inspired attention system and PCA. Training and recognition is performed by a three-layered neural network with back-propagation. The training set is composed by a huge set of indoor and outdoor images taken at the same locations under varying illumination conditions. In [52] an active stereo vision system is used to acquire appearance images. Their approach relies heavily on the training set, either for dimensionality reduction and to cut off computation complexity. Anyway the proposed framework could be easily adapted to on-line learning, provided the use of incremental algorithms.

In [36] an approach similar to our proposed has been developed. The authors concentrated mainly on the problem of robot control. They use Markov models and Gaussian mixtures for multi-hypothesis position tracking while they lack in features recognition. Furthermore the map they define is grid-like, leading to non-small position errors on the long run.

6.3 System Overview

As the robot has initially no environment knowledge, its primary goal is to build a world model. The environment is represented with a topological map, i.e. a unoriented graph where each node is a "place".

We define two different spaces, the first one being a bi-dimensional metric space $\mathcal{X} \in \Re^2$, the second one a d-dimensional features space $\mathcal{F} \in \Re^d$. $\mathcal{X}$ is the plane upon which the robot moves (we discarded the orientation, as it is not meaningful for our purpose), while $v \in \mathcal{F}$ is a features vector obtained by pre-processing a camera image. A place is defined as a subspace $P_i \subseteq \mathcal{F}$, and building an environment model is to find a partitioning of $\mathcal{F}$. Note that we are not clustering $\mathcal{F}$ (a features vector may belong to more than a place), so we need a way to distinguish between different places given a features vector; this will be accomplished by coupling $\mathcal{F}$ and $\mathcal{X}$ (see section 6.7).

A single place is modelled in a holistic fashion, where all its features vectors contribute to build its model. For each place P_i we construct a model Υ_i[2]. Such model is the probability distribution that a vector v belongs to a place P, i.e. $\Upsilon(v) \equiv p(P|v), \forall v \in \mathcal{F}$. Υ is approximated using a Gaussian mixture model

[2]In the following we will refer to a generic place as P and to a generic model as Υ, thus dropping the subscript i.

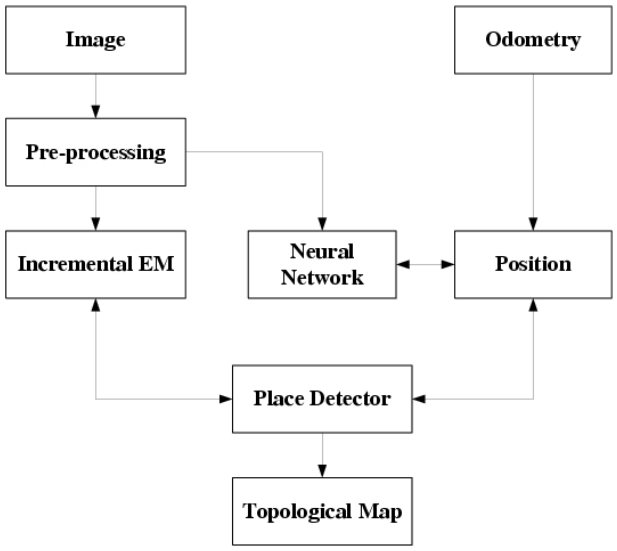


Figure 6.1: The system overview

(GMM):

$$\Upsilon = \sum_{i}^{C} \pi_i \mathcal{N}(v|\mu_i, \Sigma_i) \tag{6.1}$$

where $\mathcal{N}$ is a d-dimensional normal function with mean μ_i and covariance matrix Σ_i.

Expectation maximization algorithm (EM) gives an efficient way to compute the GMM, but it usually needs the complete training set to run. As we are building an incremental model of the environment, we will use an online version of EM; the number C of components is determined by a minimum message length approach.

Figure 6.1 shows the main components of the developed system. As the robot moves, it gathers features vectors by camera image preprocessing, updating the place model. At the same time position (in the metric space) is tracked using odometry information, and a neural network is trained to map $\mathcal{F}$ into $\mathcal{X}$.

Features and position information are combined to detect new places: if the robot has travelled for enough space and the detected features are not more modelled by Υ, then a new place is created and added to the topological map. Such process is accomplished by a place detector, which goal is to detect a new place or recognize an old one if it is already present in the topological map. The following sections will provide further details on each of the mentioned components.

6.4 Topological map

The topological map is a double oriented graph where each node is a place and an edge exists between two nodes i and j if:

$$\forall i, j \; i \longleftrightarrow j \Rightarrow \nexists k : i \longleftrightarrow k \longleftrightarrow j \tag{6.2}$$

where the operator $\longleftrightarrow$ means the existence of an edge between two nodes. Stated in other terms, two places i and j are directly linked if there is not another place k which lies between them.

Each edge is labelled with the probability $p(j|i)$ of getting from a place i to j. The probability is an uniform $1/n$, where n is the number of places directly connected to i. Note that $p(i|j) \neq p(j|i)$ because the number of outgoing connection may differ between i and j. Note moreover that $\forall i, j, k \; p(i|j) = p(i|k)$, because of the lack of a complete world frame of reference (as in absolute diktiometric maps, see [53] for example). Each place comprises a model Υ of its features, a neural network that maps $\mathcal{F} \rightarrow \mathcal{X}$, and a relative frame of reference which origin was created as soon as the robot detected the place.

From the probabilities $p(j|i)$ we can build a transition matrix $Q(i, j)$ to obtain a hidden Markov model for place transition. To avoid unrecoverable position errors the zero entries are deleted by applying a Dirichlet smoothing prior to the matrix.

The relative frame of reference attached to each node is used only during learning to compute the place centroid (see section 6.7). Its origin is placed at the point where the place was initially detected, and it is used to reset the odometry as a new learning process starts (see section 6.8).

6.5 Image preprocessing

The features vectors are computed by applying the discrete cosine transform (DCT) to images; its data independent bases allow to build the feature space incrementally. Furthermore, a useful property of the DCT is that most of the information about the signal is concentrated in just a few coefficients of the DCT [36, 54], allowing to achieve a considerable data reduction.

Each component $k = 1 \ldots d$ of the feature vector v is obtained by decomposing the image I in terms of the DCT basis functions:

$$\begin{aligned} v(k) &= \frac{2}{N} w_k \sum_{n=0}^{N-1} I(n) cos \left[k \left(n + \frac{1}{2} \right) \frac{\pi}{N} \right] \\ w_0 &= \frac{1}{2} \\ w_k &= 1, k \neq 0 \end{aligned} \tag{6.3}$$

where N is the dimension of the image treated as a vector.

For each image we retain the most significant coefficients in terms of the amount of stored information, which leads to a tractable eighty-dimensional features vectors.

6.6 Incremental-EM

For each place the robot explores a set of features vectors v are obtained by camera images pre-processing. As stated above, the probability $\Upsilon \equiv p(P|v)$ of being in a place P follows the Gaussian mixture in eq. 6.1. Usually GMM can be estimated by iterative approaches, such as Kernel methods or Expectation Maximization. We chose to adopt the EM approach, as it has been widely used in the past with success. This poses although some non-trivial issues: we do not know in advance the prior number C of components, data arrive in a continuous streaming and the model Υ has to be used concurrently with its building by the place detector.

Suppose we have a model Υ_{t-1} of the vectors obtained at time $t-1$ for the current place. The proposed Incremental-EM algorithm is stated as follows:

1. Get M features vectors from images streaming.
2. Build a model ϕ of the newly arrived data using MML approach (temporary model).
3. For each component of ϕ add it to the previous model Υ_{t-1} if and only if:
   - It has not statistically equivalent covariance with any of the component in Υ_{t-1}.
   - It has not statistically equivalent mean with any of the component in Υ_{t-1}.
4. If the above conditions are not met, the new component is merged to the one it is equivalent to.
5. Update Υ_t.

6.6.1 Building the temporary model

To explicit the dependence of Υ upon its parameters θ, we rewrite eq. 6.1 as[3]:

$$\Upsilon \equiv \Upsilon(V|\theta_i) = \sum_{i}^{C} \pi_i \mathcal{N}(V|\mu_i, \Sigma_i) \tag{6.4}$$

[3]In the following we will refer to V as a matrix $M \times d$, where M is the number of features vectors, each of them d-dimensional.

Following Shannon theory, the shortest code length for V is $\lceil -\log p(V|\theta) \rceil$. If θ is unknown the message is splitted in two parts:

$$\text{Length}(\theta, V) = \text{Length}(\theta) + \text{Length}(V|\theta) \tag{6.5}$$

which has to be minimized. This is solved by minimizing the quantity:

$$\bar{\theta} = \arg\min_{\theta} \mathcal{L}(\theta, V) \tag{6.6}$$

with:

$$\begin{aligned} \mathcal{L}(\theta, V) = \frac{N}{2} \sum_{m=1}^{C} \log\left(\frac{n\pi_m}{12}\right) + \frac{C}{2} \log \frac{M}{12} + \\ + \frac{C(N+1)}{2} - \log \Upsilon(V|\theta) \end{aligned} \tag{6.7}$$

where $N = d + d(d+1)/2$ is number of parameters of θ. The algorithm starts from an initial guess $C_{\max}$ of the components number, reducing it if a component weight became negative during iterations. (see [55] for the complete algorithm and formula derivation).

6.6.2 The complete model

Suppose we modelled the new M data with $\phi(v)$. Suppose furthermore that we already built the model Υ_{t-1} of the previous L data. Then the new model describing all the data is [56]:

$$\Upsilon_t = \frac{L}{L+M} \Upsilon_{t-1} + \frac{M}{L+M} \Phi \tag{6.8}$$

Although eq. 6.8 allows to merge historical data with newly ones, applying it straightforwardly will lead to overfitting, i.e. too much components. By testing each component k of Φ if it is statistical equivalent to any component j of Υ_{t-1} we avoid this drawback. The test is carried on in two steps: covariance equivalence and mean equivalence.

In the covariance test we determine if M features vectors $v_1, v_2, \ldots, v_M \in \Re^d$ are statistical equal to a given covariance matrix Σ_0. We first transform the data into a new vector $y = \left[\left(L_0^T\right)^{-1} V^T\right]^T$ where L_0 is a upper triangular matrix obtained by Cholesky decomposition of Σ_0. Then the test became determining if Σ_y is equivalent to the identity matrix I.

If $d > M$ (as it often happen in this application), the covariance matrix Σ_y is singular and the likelyhood-ratio test can not be performed. We can use the W-statistic defined as [57]:

$$W = \frac{1}{d} \text{tr}\left[(\Sigma_y - I)^2\right] - \frac{d}{M}\left[\frac{1}{d}\text{tr}\left(\Sigma_y\right)\right]^2 + \frac{d}{M} \tag{6.9}$$

where tr() denotes the trace of a matrix. Under the null hypothesis we have:

$$\frac{(nW-d)d}{2}+d\sim\mathcal{X}^2_{d(d+1)/2} \tag{6.10}$$

which is true as n and d go to infinity [57]..

In the mean test we determine if the M features vectors mean is statistical equal to a vector μ_0. Hotelling's T^2 statistic is well suited to this aim. It is defined as $n(\bar{x}-\mu_0)^T\Sigma_V(\bar{x}-\mu_0)$ and under the null it has distribution:

$$\frac{n-d}{d(n-1)}T^2\sim F_{d,n-d} \tag{6.11}$$

As Σ_V may be singular, we can use Σ_0 to replace it[4].

If both the tests passed, we need to merge the component k of Φ and j of Υ_{t-1}. By definition of mean and covariance we have:

$$\mu=\frac{L\pi_j\mu_j+M_k\mu_k}{L\mu_j+M_k} \tag{6.12}$$

$$\Sigma=\frac{L\pi_j\Sigma_j+M_k\Sigma_k}{L\pi_j+M_k}+\frac{L\pi_j\mu_j\mu_j^T+M_k\mu_k\mu_k^T}{L\pi_j+M_k}-\mu\mu^T \tag{6.13}$$

$$\pi=\frac{L\pi_j+M_k}{L+M} \tag{6.14}$$

where M_k is the number of data points in component k of Φ. If a component k does not have a statistical equivalent component in Υ_{t-1}, only its weight is updated:

$$\pi=\frac{M_k}{L+M} \tag{6.15}$$

while $\Sigma=\Sigma_k$ and $\mu=\mu_k$. Each remaining component j in Υ_{t-1} which has no equivalent in Φ has its weight updated:

$$\pi=\frac{L\pi_j}{L+M} \tag{6.16}$$

The model is updated finally with:

$$\Upsilon_t=\Upsilon_{t-1}\cup\{\mu,\Sigma,\pi\} \tag{6.17}$$

[4]The mean test follows the covariance test, so we already determined the statistical equivalence between Σ_V and Σ_0.

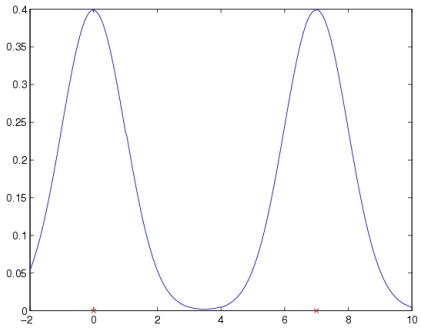


Figure 6.2: An example of two non overlapping Gaussians

6.7 From features to space

The partition of the features space $\mathcal{F}$ (see section 6.3) can not be performed using only features information, as we can incur in perceptual aliasing or overfitting, i.e. more places than needed are detected. As the robot is a mobile platform, it may be useful to integrate position information to detect the start and the end of a place. To this aim a mapping between $\mathcal{F}$ and $\mathcal{X}$ is necessary. Although the whole system may be seen as performing this mapping, we want to underline that it is computed only on a small scale (i.e. inside a single place), and an accurate position estimate is not required. In other terms such mapping is an auxiliary (although necessary) process in the context of introducing position information for place detection. In the current implementation we use a three-layered neural network to interpolate between $\mathcal{F}$ and $\mathcal{X}$, because it is easy to add new samples and it is fast to converge to good results.

The place detector needs to know where the "most representative point given Υ" is (see section 6.8). This should be the model centroid. As Υ often is composed by non-overlapping Gaussians, leading to a centroid in a zero-probability region (see figure 6.2 for a one-dimensional example), we keep all the centroids, which by definition are the mean vectors of each component of Υ.

Using the neural network we can associate to each centroid μ_i its coordinates x_i in the place reference frame. Then the place centroid (in the space $\mathcal{X}$) is:

$$x_0 = \arg\min_i \{\|x - x_i\|\} \tag{6.18}$$

where x is the robot position in the place reference frame as estimated by the odometry[5].

[5]We assume odometry being reliable for small displacements (inside a place), as it is often for

(a) First image acquired

(b) Second image acquired by rotating 180 degrees on its place

Figure 6.3: Two different images taken from the same place

Finally the distance between the robot and the centroid is modeled by a normal bivariate distribution with spherical covariance $\Sigma = \sigma I$ to take into account of odometry errors:

$$\text{pos}(x|\Upsilon) = \frac{1}{(2\pi)|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - x_0)^T \Sigma^{-1}(x - x_0)\right) \tag{6.19}$$

6.8 Place detection

Suppose the robot is constructing a new model of a place, let it be a room with some windows along one of the walls. Suppose furthermore that at the beginning it is facing a wall and next it turns 180 degrees without traslating, facing the wall with the windows (see figure 6.3 for an example). It would be very hard for an unsupervised pattern classification algorithm to cluster together features gained from the first and the second image. But if we integrate position information (only the (x, y) pair), the robot should be able to recognize that the place has not changed.

This simple example is useful to introduce the last component of our system: the place detector. Its goal is to detect, during learning, when a place has changed so that we need to create a new model, or recognize a old one. Informally, we have a new place if the features the robot is gathering are not more modelled by the current model, while at the same time the robot has moved far away the place centroid.

---

most modern odometry systems.

Formally, the robot detects a new place if the product of the likelyhood of the current features vector v given the place model Υ times the distance between the robot and the place centroid (as in eq. 6.19) is above a threshold δ_n:

$$\{\Upsilon(v) \times \text{pos}(x|\Upsilon)\} > \delta_n \tag{6.20}$$

If the robot detects a new place then a new node is added to the topological map, the odometry is resetted to zero and a new frame of reference centered at the current position is added to the new place.

If the topological map is not empty, the robot tries to recognize a place before adding a new one. This is performed by checking that the max likelyhood of the current features vector under any known place model, weighted with the transition probability, is greater than a threshold. Formally if the robot is coming from a place P_j, then a place P_i is recognized if:

$$\left\{\max_i \left[\Upsilon_i(v) \times Q(i|j)\right]\right\} > \delta_r \tag{6.21}$$

When this happens the model Υ_i is updated using the proposed algorithms, as if it was just created; otherwise a new place has to be added.

6.9 Experimental results

We performed two separate tests, the first one involving only the feature detector in a standard appearance-based navigation fashion, the second one involving the complete system.

For the first test we manually controlled the robot along one corridor taking 360 images from both the stereo cameras every 30cm. The left camera odd images have been used for the training set T, the right camera even images for the validation set V. The training was simply performed by appling the DCT transform of eq. 6.3 to all the images in the training set. The same process was applied to all the images in the validation set. For each vector i in the validation set we chose the one closest to a vector j in the training set using the formula:

$$j = \arg\min_{k \in V} \|V_i - T_k\| \tag{6.22}$$

where V_i and T_k are the d-dimensional rows of matrix V and T. This way we achieve a position-independent error estimate. The plot of all the (i, j) pairs is shown in figure 6.4; we plotted a red line with all the correct pairs (i, i) as a reference. As it can be seen the error is too large and too frequent to make a correct localization.

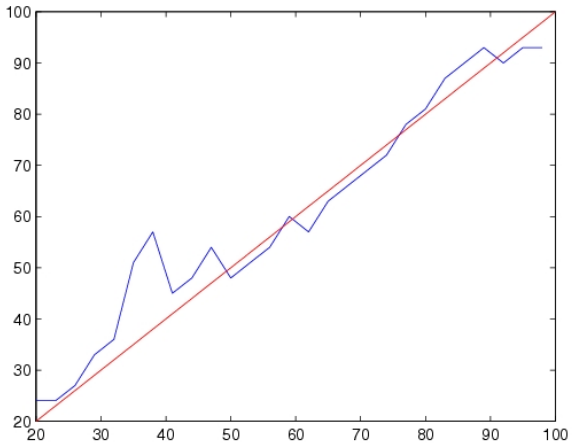


Figure 6.4: The first test results.

The second test was performed by letting the robot move autonomously. We provided a set of simple low-level behaviors to explore our laboratory environment, composed by a room and two corridors. The test has been repeated 20 times, performing nearly 100 place detection and recognition processes per place. The robot partitioned the environment in four places, which we identified as a room and three corridors (actually the third corridor was located at about the junction between the two real corridors). Results are shown in figure 6.5. On the x-axis the place identifier is reported. For each place $p_i, i = 1 \dots 4$ we plotted how many times the robot detected the place $p_j, j = 1 \dots 4$. The first place was correctly recognized 80% of all the trials, the second one 90%, the third one 75% while the last one 95%, achieving a good overall performance.

6.10 Conclusions

We have introduced a robotic system able to simultaneously build a topological map of the environment and to use this map for localization. The base definition we introduced is "place", meaning a set of features which share a common representation and are close in the space. Places are connected together to form a topological map, also the base for a hidden Markov model. Furthermore position is integrated with features detection for the purpose of place recognition.

Our main concern was about building an incremental model of the environment as the robot starts without any prior knowledge. This leaded to the adoptions of several statistical incremental learning algorithms, which proved to be as reliable

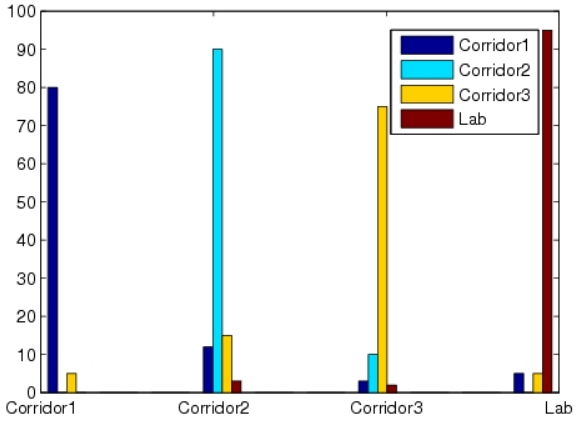


Figure 6.5: The second test results.

as the batch ones.

In this chapter we introduced a novel methodology for on-line appearance based navigation. Although we did not use the proposed system in the context of robotic museum guides, the concepts illustrated here will form the basis for chapter 7, where appearance based navigation is heavily used in the context of node recognition.

Chapter 7

Towards Topological Maps

7.1 Introduction

In chapter 5 we showed the problems that arise when moving from a small structured indoor environment to a large outdoor one. Many of them were solved using the GPS and confining the metric mapping to a local scope where precision was needed. In some applications GPS data may not be available, or they could be discontinuous or too noisy (like in urban or cluttered environments). In this chapter we will develop a mapping system based only on topological information, thus eliminating the need of any metric environment representation. The maps generated this way may be used for high-level motion planning, like road planning in urban environments, and metric information may be added where the robot task needs precise position information. We will use only camera images, as a mean to investigate vision based SLAM. Part of this work is based on the material in chapter 6, namely the appearance node recognition sub-system. Although the proposed system is usable either in indoor and outdoor environments, it is more suited to large ones where there is not too much clutter (see section 7.9.

Vision based Simultaneous localization and mapping (SLAM) is currently one the major challenges for robotic research community. While noteworthy results has been achieved using range finders (laser or even sonar), vision still poses challenges. These are mainly related to the burden of computational complexity, because the data stream is much larger compared to range sensors. Furthermore vision is mainly affected by illumination changes and noise, for instance in the case of poor light conditions or strong contrast. However such large stream of data produced by cameras may be exploited to solve problems that are still hard for range finders, like for example perceptual aliasing or data association.

Our work will focus on large semi-structured environments, i.e. environments which human intervention has not radically modified, like office buildings and

similar. In particular we assume that navigable space is clearly distinguishable from obstacles, and it should be laterally bounded in space, as in the case of roads or pathways. Examples of semi-structured environments include parks, urban roads or even large buildings. We carried on experiments in the botanical garden presented in chapter 5, as an instance of a large semi-structured environment, but results has been obtained also in an office-like environment.

Our primary goal is to build an environment representation suitable for localization and path planning. Such representation should be flexible enough to account for sensing and action uncertainty, allowing it to be robustly modified as new information is added or the old one is updated. Furthermore the entire process should be autonomous and unsupervised, denying then the use of off-line training procedures or previously collected databases. The most suitable choice for our purposes is a topological map. Although some works exist that try to merge the two metric and topological maps (see for example [58]), using a metric map usually involves computing a description of the environment which order of precision is often the centimeter. Topological maps on the other side often involve the use of landmarks, i.e. features of the environment that are easy to locate and distinguishable.

We define a topological map where each node describes a characteristic of the environment, namely the presence of a branching in the navigable space, without any landmark position or distance information. This is mainly due to the poor odometry of our robotic platform, but it is supported by the requirement of the map to be scalable to large environments. A great effort has been applied to obtain a robust node detector, because it will be the basic building block of our architecture. Furthermore in order to deal with action and sensing uncertainty we use a Bayesian framework to keep track of the robot belief. The same framewok is used to augment the topological map with uncertainty measures, so that it is easily modifiable when new observations are gathered.

7.2 Related Works

This work has been mainly inspired by [59], where a hierarchical representation of the environment is proposed. The topological layer comprises nodes placed at distinctive places. The definition of distinctive places is mainly sensor related, i.e. a place needs to be easily distinguishable from other ones given only sensory information. However this approach relies on a complete observability of the robot state, assumption with is easily violated when working with real robotic platforms.

The SLAM problem concerns estimating a map of the environment while the robot pose is unknown, using sensory and controls information. Probabilistic algorithms proved to be reliable enough for the task (see for example [6, 60]. Two

fundamental mainstreams for solving the generic SLAM problem are the Extended Kalman Filter (EKF) [61] and the Rao-Blackwellised Particle Filter (RBPF) [62]. The first one has been widely adopted due to its simple implementation and effectiveness. Its major drawback is its poor scalability because the covariance matrix cannot be efficiently updated as the number of landmarks increases too much. Many efforts have been dedicated to enhance the efficiency of the EKF-SLAM, like factorizing map information [35]. On the other hand, the RBPF SLAM approach exploits the conditional independence of landmark measurements given the robot trajectory, while maintaining a multi-modal posterior distribution. Impressive results have been achieved using laser range finders sensors to create metric maps [30, 63].

Probabilistic approaches for SLAM have been applied for the creation either of metric maps and topological maps. Some works recently tried to merge metric and topological maps, for instance Poncela et al. [58] use a hierarchical subsampling of metric space to obtain a coarse topological map. A different approach is in [64], where a data driven learning of hidden Markov models is used to build a topological map. Few works have addressed the issues related to inference about topological maps. In [49] the space of topological maps is analyzed to obtain a map closely related to collected data. Markov chain Monte Carlo methods are used to sample this huge space and evaluate the likelyhood of topological map hypotheses. This approach has still to be proved scalable to large environment.

Several vision-based SLAM approaches have been proposed which use either EKF or RBPF. A major part of the work extracts 3D landmarks based on image features. Faugeras et al. [65] used an EKF both to create a 3D map composed by line segments and to localize the robot by registering the local map into the global one. SIFT [66] point features have been applied to the SLAM problem to extract 3D landmarks [67]. In this system, odometry data provide a rough estimate of the robot motion that is used to predict landmarks locations in the next image. Then SIFT features are matched and a least-square approach is applied to refine both the robot pose estimate and the map. This approach has been extended [68] to deal with global consistency. Local submaps are merged in a global map and a loop closure constraint is applied to minimize the effect of the accumulated drift.

Sim et al. [69] also use SIFT features combined with an RBPF. Their approach does not use odometry data, relying on visual odometry to compute the robot motion model. The approach proposed in [70] is based on the EKF and relies only on visual information provided from a single camera to solve the SLAM problem. Image regions detected using the saliency operator of Shi and Tomasi [71] are used as landmarks. A similar approach is in [72] where a stereo camera is used to compute the visual odometry, while landmarks are extracted using the Harris corner detector [73].

A topic closely related to visual SLAM is visual navigation. The main problem

is to find a map of navigable space given image information (usually stereo images). The map is represented as an elevation map or traversability map, where each cell contains information about the elevation of that small region or cost-to-traverse. In [74] stereo vision is used to obtain an elevation map used for path planning. A close work is presented in [75] where color information is coupled with range information to obtain an obstacle map. In [76] the problem of finding road junctions has been addressed. However this application is limited with recognition of urban roads, being no useful for general kind of junctions (a problem we try to address in this work).

A slightly different approach to visual SLAM relies on appearance based navigation. Appearance based approaches usually need to train the system off-line with images describing all the environment often under varying conditions. This is usually due to the need of input data dimensionality reduction, in most cases performed by principal component analysis (PCA). In [50] the system is trained with features vectors built from a large database of images taken either in indoor and outdoor environment. Recognition is aided by a hidden Markov model for transition between places. In [51] features vectors are extracted using an underlying biological-inspired attention system and PCA. The training set is composed by a huge set of indoor and outdoor images taken at the same locations under varying illumination conditions. In [36] Markov models and Gaussian mixtures are applied for multi-hypothesis position tracking. In [77] an on-line version of appearance based place detection is proposed where a place detector decides which features belong to which place integrating odometric information and a hidden Markov model. An appearance based approach is adopted in [78] to detect loop closure; 3D SLAM is performed using a 3D laser scanner.

The main contribution of this work is a novel mapping algorithm based mainly on visual information. By introducing a functional concept of node, the robot creates maps that closely resemble roadmaps. Fur themore dealing with large environments with a robotic platform affected by poor odometry induced us to define a topological map which is no metric related. This means that there are no landmarks and no metric information to be estimated from the environment. Although this approach denies the possibility of having the robot pose known with precision, we hypothesize that this is not the main goal in large environments applications. Nevertheless our approach allows merging metric and topological maps for instance by augmenting each node with a small scale detailed map, where needed.

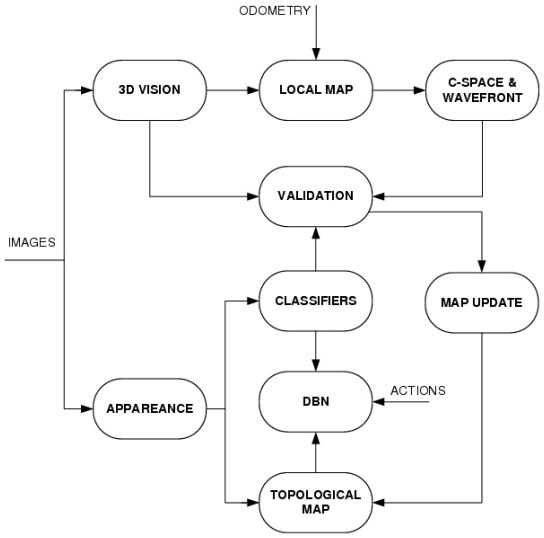


Figure 7.1: An outline of the proposed architecture.

7.3 Overview

The proposed system performs exploration and SLAM at the same time. Although the exploration process is not detailed here, it should be pointed out that it is a crucial step for the map building and updating. Going through the same nodes and edges more and more times will produce a more correct and coherent map. In this section we will overview the main components of our system, outlined in Fig. 7.1.

The robot goal is to build a topological map describing the choices the environment offers in terms of branching in the navigable space. The topological map is a undirected graph; each node is a place in the environment where the robot may follow more than one path. In order to discover such places a local metric map of the environment has to be built. Although this may seem in contrast with our approach, it is needed in order to exploit the environment structure and to find navigable space.

A local map is a detailed metric map built by performing 3D reconstruction of stereo images and by projecting to a 2D plane the obstacles. Even though the compression of the dimensionality results in a loss of possible recognizable features (e.g. shape), the remaining information is sufficient for the subsequent steps to be performed. Navigable space between obstacles will represent a possible branching of the pathway.

The local map is very bounded in space, as it relies on odometric and visual information, which are well-known to be very noisy. Namely 3D reconstruction is

performed using a maximum range threshold of 10 meters, as the performances of the stereo reconstruction are usually unreliable at longer ranges. Moreover we consider robot displacement of at most 3 meters, as experimentally it has been noted that our robotic platform odometry is unreliable for longer distances.

The local metric map is represented as a 2D occupancy grid map [19] where each cell contains the probability of being occupied. In contrast with previous approaches we do not need an accurate representation of the environment from a metric point of view. As a consequence, inaccuracies due to stereo mismatch do not affect the proposed approach. Subsequent steps will take care of detecting false positive, i.e. the absence of navigable space between obstacles. Even though some obstacles regions in the local map are sparse and noisy, the incremental building of the occupancy grid results in the filling of the "holes" between adjacent obstacles. Moreover by accumulating data over time we overcome the problems due to the limited field of view of the stereo camera.

When a new node is discovered, it needs to be validated, i.e. each potential branching should be checked to be sure it is not due to reconstruction errors. This is accomplished by moving the robot to an observation point and checking if there is enough space to account for navigability. The second step is to memorize the appearance of each edge, information needed either for localization and navigation purposes. Appearance based localization and navigation is a growing topic that has shown good results even in large unstructured environments [50, 77, 47, 36, 51]. Moreover it is not very susceptible to perceptual aliasing, as it uses the whole image as data to learn and to recognize. Using the sum of appearances of edges allows us to reduce the node recognition problem to smaller classification problems, easier to solve and lesser susceptible to perceptual aliasing. Furthermore the robot builds a probabilistic model of the edges appearance that is useful to integrate observations during localization.

Localization is performed using Bayesian recursive integration of beliefs with actions and observations. As the robot real position is unknown, multiple hypothesis have to be tracked. This means that we have a discrete probability distribution (belief function) over the map nodes. Each time the robot performs an action, i.e. each time a robot travels over an edge, and each time it gather observations, i.e. it checks the appearance of a node, the belief needs to be updated. This is accomplished by performing exact inference in a dynamic Bayesian network unrolled for two time slices.

Map updating is a crucial step for the success of the proposed system. As every information gathered by the robot is corrupted by noise, the topological structure of the map should be flexible enough to allow changes coherent with the new environment discoveries. The only information which will be fixed in time during map building and updating is the presence and appearance of nodes. This means that once the discovery and validation of a node terminates, the node will never be

deleted from the map. Although this may seem a strong assumption, experimental results showed that nodes identification is reliable and robust to noise. The edges connecting the nodes may vary instead. This may happen if the robot started from a node i, followed an edge $e_{i,j}$ but reached a node $k \neq j$. In this case various options may be considered, namely that the robot was not in the node i, or it is not in node k, or the edge $e_{i,j}$ was wrong. Depending on the belief and the edges "credibility" this may lead in the worst case to edge deletion or creation of a new one.

Navigation is performed by following an edge until the next node is detected. An edge is not required to be a straight connection between two places, but navigating between two nodes is a straightforward process not error prone. This follows from the node definition: navigating an edge could be related to travelling through a corridor, without any multiple choice. If a branching is detected, then it must be a node. Furthermore the robot can choose among the edges by using their appearance.

Next sections will explain in more details the introduced architecture modules.

7.4 Node discovery

Node discovery is performed by constructing a short-time local metric map integrating the visual information the robot sensed in a bounded region. Each time a region shows potential multiple choices, a node may be created[1].

Each time step the robot performs a 3D reconstruction using the Triclops stereo vision module (see 7.9). The result of the stereo procedure is a *point cloud* referred to a local frame of reference located in the right camera. A Hough transform is used to estimate the ground plane in front of the robot [79]. The 3D points are then classified in floor points which are not considered for updating the map, and obstacle points. The local map is updated by projecting down the obstacle points and pruning away the points too high to represent a real obstacle for the robot (e.g. the tree's branches overhanging the road).

The local map is thresholded to obtain an obstacles binary image. The pixels are then clustered using the connected components algorithm [80] and the too small clusters are discarded. The obstacles image is converted in a space of configurations for the robot using the star algorithm [81]; the result is still an obstacle binary image. The connected components algorithm is applied again to obtain a set of disjoined obstacles. This two steps procedure allows to obtain the robot navigable space and to minimize the impact of the 3D reconstruction errors. A modification

[1]Due to errors in the visual processing, a potential choice has to be verified prior creating a node, as explained below.

of the wavefront algorithm is used to compute the minimum distances between each pair of obstacles:

1. For each pair of obstacles (O_i, O_{i+1}), starting from the lower left and in a clockwise pattern:
2. Expand the obstacles using the wavefront until the two waves collide in some point P_i.
3. Select the next pair of obstacles (O_{i+1}, O_{i+2}) and go back to step 1.

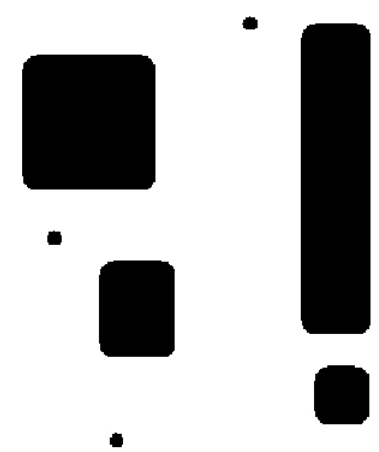
This way we obtain a list of points P_i, each of them representing a possible path choice (edge). If there exists more than one such point then a node could be created.

Before adding a new node to the map, it has to be verified that either the selected points really span a new path (*clearance check*) and the node is not already present in the map (*memory check*). The robot needs to find an observation point from where to look at all the extracted points P_i. The observation point is computed as follow:

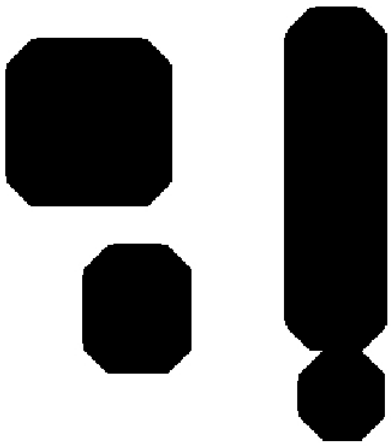
1. Select among the set P_i the farthest point as goal g (using the robot position).
2. Use the wavefront naive algorithm to obtain the shortest path between the robot and g.
3. Simulate the robot following the path.
4. For each step:
5. Apply the Line Sweep algorithm [82] to the P_i set and the current position.
6. If all the points are visible, then return the current position V.
7. Go back to step 5.

According to the Line Sweep algorithm, it is possible to obtain a clear view of all the P_i from V. The robot then moves to the observation point V, it uses the pan-tilt to look at each point P_i and performs both the checks named above.

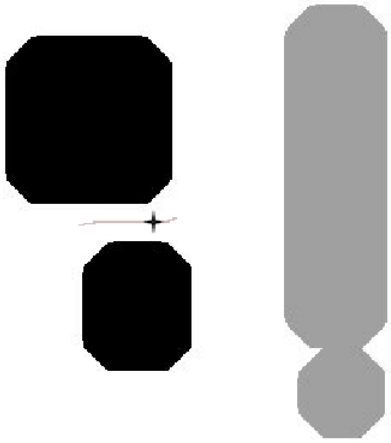
The above procedure is illustrated with a toy example in Fig. 7.2. In Fig. 7.2a the robot has built a local binary map composed by four large obstacle plus some noise. After applying the first clustering and removing small clusters the noise is removed. The second step (Fig. 7.2b) is to create the configuration space, which has the effect of connecting the two obstacles on the right into a larger one. Next the minimum distance between pairs of obstacles is computed by iterating between pairs of obstacles in a clockwise way. In Fig. 7.2c the first distance is computed,



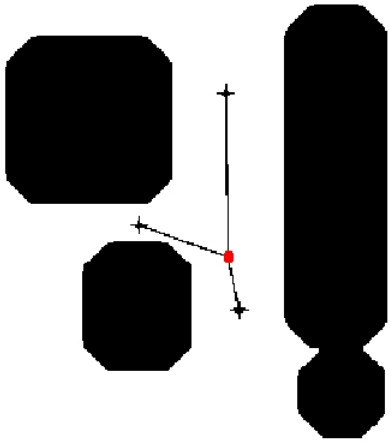
(a) Initial conditions



(b) After pruning small clusters and creating the C-Space.



(c) After the wavefront expansion.



(d) An observation point is found.

Figure 7.2: Illustration of the node discovery procedure. See text for details.

Figure 7.3: Local map shifting. As the robot moves (from the red point to the green one) the local map shifts according to the new perceptions.

and the first point of contact between the two wavefronts is shown as a diamond. The final step is to find an observation point, shown as the red one in Fig. 7.2d.

The memory check is described in section 7.7. The clearance check is performed by a weighted sum of the number of points belonging to the ground floor, weighting more the fartest points. Only the points P_i that pass both the checks are retained. If the remaining points still number more than one, then a node is added to the topological map.

The local map is updated until the robot has traveled for more than 3 meters. In fact, below this threshold the odometry estimate can be considered sufficiently accurate for our purposes and the resulting local map is piecewise correct. The local map can be considered as a dynamic window spanning a fictitious global map. As the robot moves new pieces of information are added to the local map while old ones are discarded (see Fig. 7.3). As a consequence past data are not completely erased as two subsequent local maps are partially overlapped. This allows to reduce the number of false negative due to the limited field of view of the stereo camera.

Furthermore if a possible branching has been detected and the robot is approaching the corresponding observation point, then the local map updating is suspended.

7.5 Map Building and Updating

In order to manage a topological map we need to update nodes and edges as the robot gathers new information about the environment. The only piece of information upon which the robot can rely is the presence of nodes, as the discovering and validation process described in section 7.4 assures us that a node is really present. What we do not know is where the robot really is and, more important for a SLAM application, what is the map topology. The first kind of uncertainty is addressed by the use of Bayesian integration of the belief function, as described in section 7.7. This allow us to know, with a great degree of confidence, in which nodes the robot is most likely to be. The second source of uncertainty arises from the

edges linking the nodes. In order to link two nodes we must be sure of how many edges may depart from a node (this is a reliable information), and how reliable is the information that the robot was in node i before arriving in a node j. We can use again the belief function to obtain a degree of confidence that an edge really exists. This is exploited in the following, where we propose a novel algorithm to manage the map, allowing edges to be created or deleted as the robot became more confident in its beliefs.

The environment map is represented as an unoriented graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. Each edge e has a value $c(e)$ associated representing its "credibility", i.e. the probability that the edge really exists. A special kind of edge is the *dangling edge*, i.e. an edge that does not connect to a node: future exploration will replace a dangling edge with a real one.

When the robot detects a new node (after both memory and clearance checks), this has to be connected to the existing ones. All the active nodes in the belief function, i.e. $\{i|P(x = i) > 0\}$ are eligible to be linked to the new node. On the other side, if the robot detects an old node, all the active nodes in the previous belief could connect to all the active nodes in the new belief. This leads to a competitive node linking process, where each node tries to gather resources represented as the other nodes edges. This competition is regulated by the value $\phi(x_{t-i}, x_t) = P(x_{t-i} = i)P(x_t = j)$, which represents the degree of confidence that the robot travelled from node i to node j. Obviously if a node tries to connect to more than one node using the same edge (i.e. by performing the same action) then it will choose to create only the connection with the highest ϕ value. The following procedure is used to connect node i to node j, to be applied either to new node or existing ones:

- While there are not more nodes to be connected:
- **Connecting dangling edges**: If the robot was going through a dangling edge belonging to node i, and node j has a spare dangling edge, then create a new edge between the two nodes and set its value to ϕ.
- **Competition between nodes**: If the robot was going through a dangling edge belonging to node i, and node j has not a spare dangling edge, then:
  - If ϕ is greater than the edge in j with lower $c(e)$ then delete e and create a new edge (i, j).
  - Otherwise the new edge is not created.
- If the robot was going through an existing edge e then:
  - **Increasing the edge value**: If e is already connecting i and j, then replace its value by $\max\{c(e), \phi\}$.

- If e is connecting another edge $k \neq j$, then:
  - **The new edge is more credible than the old one**: If $\phi > c(e)$ then delete e and create a new edge between i and j, setting its value to ϕ.
  - **No map updating**: Otherwise do not connect the two nodes.

Although this procedure may lead to the creation of *phantom edges* (i.e. a link between two nodes that are not really connected in the environment), further explorations will lead to refining the edges as the robot become more confident in its position. Furthermore it should be noted that the observations $P(y|x)$ are not included in the edge values, as they are already included in the belief calculation, otherwise we will count them twice.

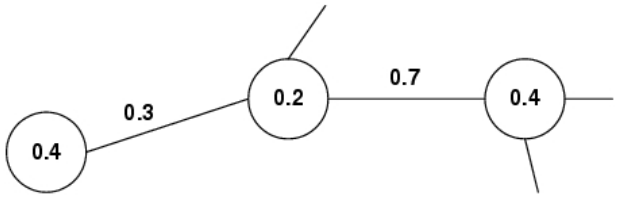
An example of this procedure is shown in Fig. 7.4. The start condition is shown in Fig. 7.4a, where the topological map has three nodes and the belief is $P(X) = \{0.4, 0.2, 0.4\}$; fur themore two edges are labelled with their value, whereas the other edges are dangling ones. The robot chose three actions, one for each node with belief greater than zero, shown in Fig. 7.4b as the red edges. A new node with three edges is discovered and validated; the edge from which the robot was coming from is shown in red. The map is then updated, and the result is shown in Fig. 7.4c. Here two new edges are created, linking spare dangling edges, and labeled with the belief of the tail node. The first node had no dangling edges, but the one the robot was following had a lower value than the new one. This means that the new edge is more "credible" than the old one, and it can be replaced, leading to the new map configuration shown in Fig. 7.4c.

7.6 Node Appearance

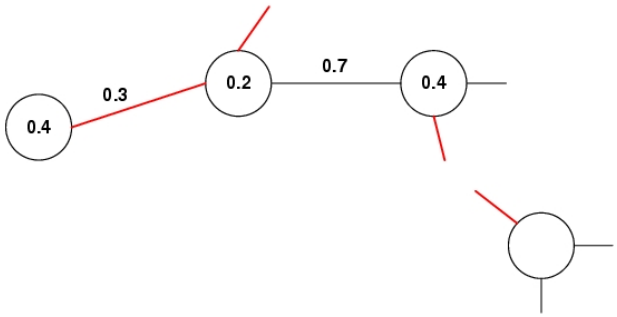
The robot needs to memorize a node appearance to recognize it in the future. We define the appearance of the node as the appearances of all its edges, so that to recognize a node the robot needs to check the appearance of all the edges. When a new node is added to the topological map, the appearance of each new edge is obtained from the observation point. We follow a procedure similar to the one in [77], where the appearance of the edge connecting node i to j is modelled as a Gaussians mixture model:

$$\Upsilon_{i,j} = \sum_{k}^{C} \pi_i \mathcal{N}(v_{i,j}|\mu_k, \Sigma_k) \tag{7.1}$$

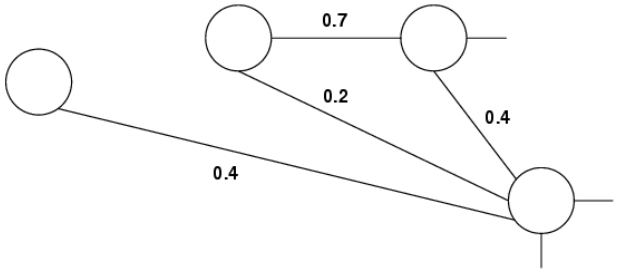
The features vector $v_{i,j}$ is extracted from the images using Principal component analysis [47]. For each edge a bunch of images is collected by moving the robot



(a) The initial graph.



(b) After discovering a new node.



(c) Map updating.

Figure 7.4: An example of map updating. See text for details.

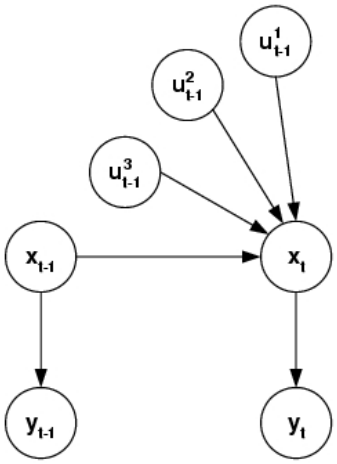


Figure 7.5: The Dynamic Bayesian Network used for localization unrolled for two time slices. In this example the map comprises three nodes, therefore there are three action nodes u^i_{t-1}

pan-tilt unit. Each feature vector is reduced to a low-dimensional one via PCA. Learning is performed using MML-EM [55] to produce the model in eq. 7.1.

Once the training is performed we obtain a sensor model $P(e_{i,j}|v_t)$ representing the probability of sensing the edge $e_{i,j}$ given the features vector v_t. This model will be used for localization purpose (see sec. 7.7).

7.7 Localization

The localization problem concerns calculating the probability $p(x_{1:t}|y_{0:t}, u_{1:t})$ of going through the states $x_{0:t}$ given the past observations y and controls u. This is accomplished by the well-known Bayesian recursive formula (discrete case):

$$p(x_{1:t}|y_{0:t}, u_{1:t}) = p(y_t|x_t) \sum_{x_{t-1}} p(x_t|u_{t-1}), x_{t-1})p(x_{t-1}) \tag{7.2}$$

graphically represented by the Dynamic Bayesian Network in Fig. 7.5.

According to our model the robot state is a discrete probability distribution (referred to as *belief* function b_t) over the topological map nodes, i.e.:

$$P(x_t) \equiv P(x_t = i), i = 1 \ldots |V| \tag{7.3}$$

To avoid unneeded wasteful computations we delete states with a very low probability, normalizing the resulting belief function. The observation model $P(y_t|x_t)$

is derived from the node appearance in eq 7.1, assuming that the appearance of every single edge is independent of all the other appearances:

$$P(y_t|x_t = i) = \prod_{k \in edg(i)} \Upsilon_{i,k} \tag{7.4}$$

where $edg(i)$ is the set of all the edges of node i.

The set of possible actions is dependent on the current node, i.e. $U \equiv U(x)$, and it represents going along one of the edges of the node. The set of all the possible actions given a belief b_t is the union of all the possible actions:

$$U(b_t) \equiv \bigcup_{x|P(x)>0} U(x) \tag{7.5}$$

This model allows us to split the set of all the possible actions to several smaller sets of actions each of them depending on the node. During navigation the robot has to choose which action to perform, i.e. which edge to follow, for each node which a probability greater than zero (note that nodes with a small probability are not considered, as explained above). The transition probability is then modelled as:

$$P(x_t = i|x_{t-1}, u_{t-1}) = \begin{cases} 0.9 & \text{if } u_{t-1} = \text{goto } i \\ 0 \text{ if } xt - 1 = x & \\ \frac{0.1}{|edg(i)|-1} & \text{otherwise} \end{cases} \tag{7.6}$$

To take into account errors due to odometry or perceptual aliasing we induced a small error in the transition, dependent on the number of edges of a node. The second case in equation 7.6 is needed to avoid some perceptual aliasing problems: it is assumed impossible that a robot starts from a node, follows an edge and then finish in the same node it was starting from. This means also that it will be impossible for the map updating process to add an edge connecting a node to itself, as navigating an edge means moving from one node to another one.

Localization is performed by unrolling the DBN for two slices and then applying the junction tree algorithm [83], treating the observation and control nodes as evidence. Each time the robot recognizes the presence of a node (as in section 7.4) it moves to the observation point and for each potential edge it queries all the nodes appearance using eq. 7.4. If the result is grater than a threshold for some node j, then the new node is recognized as an old one and the belief is updated as described above. If no node is recognized, a new one is added to the topological map and the belief is updated including the larger state space. It may happens during map updating that an edge is removed and a new one is added. In this case the action associated with the old edge is replaced with the new one prior updating the belief.

7.8 Navigation

Although navigation is not a full covered topic in this chapter, we will outline it in an informal view. Imagine the robot having a plan to go from node i to node k. If we assume that the state and actions are fully observable (otherwise we should use a partially observable Markov decision process, see section 7.10), a plan may be generated using a graph search algorithm like Dijkstra algorithm or A*. Furthermore we could augment each edge with its length (although this information is very subject to odometric errors) or time-to-traverse, to obtain an optimal path.

Once a plan has been generated, navigation is performed by identifying the edges that connect each node in the plan. This is accomplished by moving the robot in the observation point as explained in section 7.4, and checking the edge appearance with the one needed to reach the next node. Then the robot may start following an edge until it detects a new node, repeating the process again.

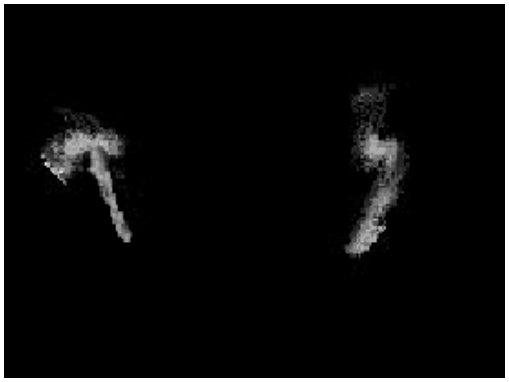
Edge following may be accomplished by a low-level routine like corridor following. Although we do not have necessarily a corridor to follow, we could rely on the fact that no fork may happen during edge traversal, unless the robot arrived in a node. This means that in order to navigate an edge the robot needs only to navigate the free space going away from the starting node until it finds a new one. This is somewhat the same process we human could use when navigating in an urban environment or along highways.

7.9 Experimental Results

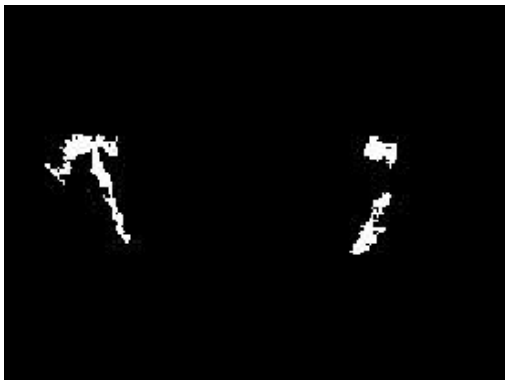
The robotic platform used for our experiments is a Pioneer 3-AT from Mobile Robots. The robot is equipped with a Bumblebee2 stereo vision camera that is mounted atop a pan-tilt unit and it is connected via an IEEE 1394 link to an EBX form-factor single board PC, powered by a Pentium Mobile processor clocked at 1.6GHz. The Triclops Stereo SDK included with the Bumblebee2 performs Sum of Absolute Differences (SAD) stereo correlation and produces dense disparity images. Furthermore the robot is equipped with a laser rangefinder and a GPS unit, although we did not use them for our experiments.

Experiments were conducted in the context of a botanical garden robotic guide [84]. The environment is characterized with long pathways sided with walls or plants. The terrain is often covered by foilage and sometimes muddish. This description fits our definition of semi-structured environment, because the pathways are screened by obstacles and connected by junctions or large plazas. The robot was teleoperated while moving along pathways, but switched to automatic control when it detected a node. The route was about $500m$ long.

(a) Rectified image.



(b) Occupancy grid after 1m travelling.

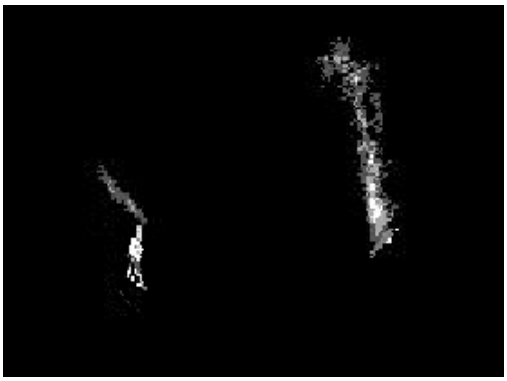


(c) Binary obstacle image.

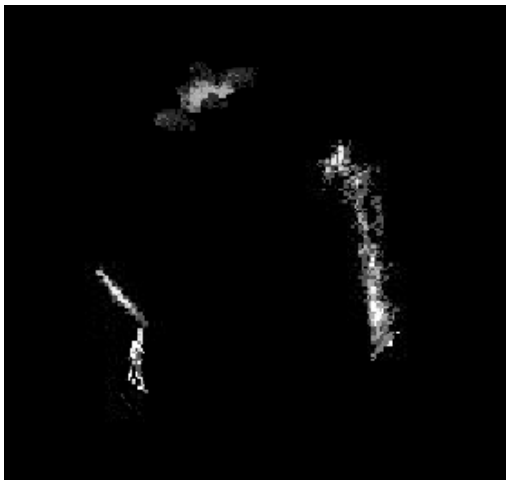
(d) Configuration space.

Figure 7.6: A local map without detecting nodes.

(a) Rectified image.

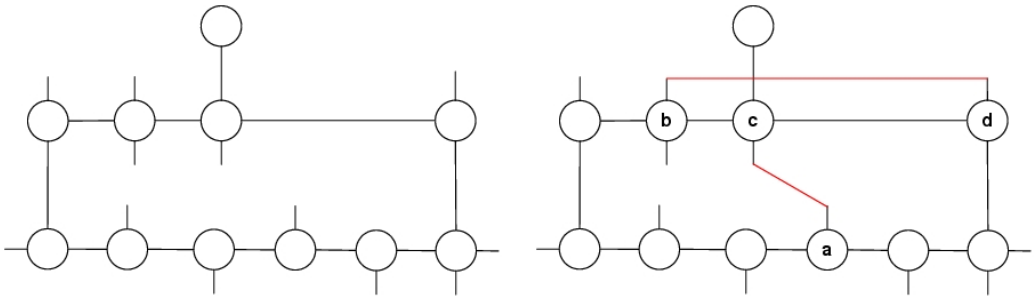


(b) Occupancy grid after 1m travelling.



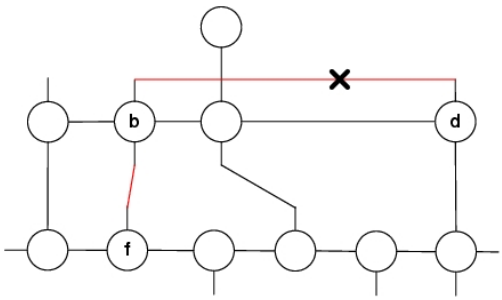
(c) Occupancy grid after 2.5m travelling.

Figure 7.7: A local map. A potential node has been detected

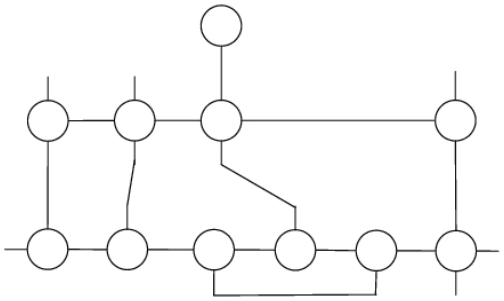


(a) Topological map built after closing one loop.

(b) Creation of a phantom edge.



(c) Correcting the topological map.



(d) The final map.

Figure 7.8: Steps in creation of the topological map.

Two snapshot taken along the route are shown in Fig.s 7.6 and 7.7. The first one was taken along a pathway where the robot had no choice but going forward. Fig. 7.6a shows the occupancy grid built after $1m$ travelling. The corresponding binary obstacle image (Fig. 7.6b) shows that reconstruction errors leaded to a phantom hole in the right obstacle. This was corrected by the star algorithm during the configuration space creation (7.6c), where the two obstacles on the right were merged to became only one. The second snapshot corresponds to a junction near a fountain (Fig. 7.7). In Fig. 7.7b the junction was too far away to be detected, but after $1.5m$ travelling the robot sensed the trees in front of it detecting a potential node (Fig. 7.7c). Subsequent validation leaded to the creation of a new node.

The robot was allowed to make several runs in the environment in order to validate the map updating algorithm. During the first run the robot discovered all the nodes in the map. Several phantom nodes were created along a pathway where small pathways in the side walls where discovered. However they were too small for the robot to traverse, and they were discarded during the validation process. The result of the first run is shown in Fig. 7.8a. All the edges and nodes were correctly discovered, although many internal links were not yet discovered. Several runs

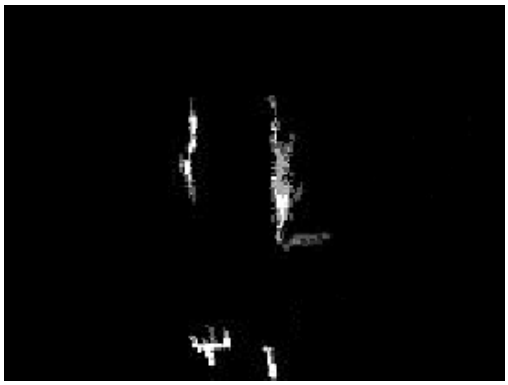
were performed to discover the inter-connections between nodes. The updated map after the first run is shown in Fig. 7.8b: here the presence of multiple unexplored choices (dangling edges) leaded to the creation of a phantom edge between nodes b and d. Subsequent runs deleted the edge (b, d) to create the right edge (b, f). The final map is shown in Fig. 7.8d: it took 11 runs to create it, with a total travelling distance of 2.3 km.

In order to further validate our approach the system was tested in an indoor environment. The robotic platform we used is a Performance PeopleBot from Mobile Robots, equipped the same way as the P3-AT. Fig. 7.9b shows a correct node detection. Even if three branches were detected, the one on the left was correctly discarded after validation, as the robot became rid of the door occlusion. Fig. 7.9d shows a false negative: the robot was not able to detect the open door at the end of the room in Fig. 7.9c. A more careful environment exploration would have surely discovered it, but the assumption of large semi-structured environment conflicted with the cluttered image presented in Fig..

7.10 Conclusions

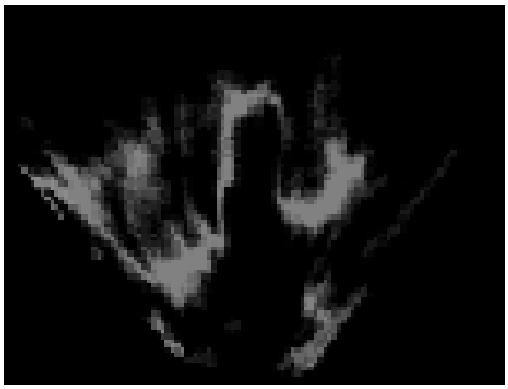
In this chapter we presented a novel approach to simultaneous mapping and localization based mainly on visual information. The main contribution is the functional definition of a node, i.e. as a place where a branching of the navigational space appears. Furthermore the robot is able to build a map which is in no way metric related, thus it is not affected by position estimation errors. Metric information is used only to build a local map, which is small enough to be unaffected by odometric or 3D reconstruction errors, and it is needed only to place detection purposes. Bayesian inference is used either for localization and map updating, to take into account of sensing and actions uncertainty. Data aliasing is addressed by using multiple instances of images appearance and integrated in the recursive belief updating. Finally experimental results showed that the proposed map updating algorithm is strong enough to deal with loop closures problems, although it needs several environment exploration runs to create a coherent map.

(a) Rectified image.



(b) Occupancy grid after 0.5m travelling.

(c) Rectified image.



(d) Occupancy grid after 0.5m travelling.

Figure 7.9: Local maps in indoor environment.

Part IV

Conclusions and future works

One of the main goal of robotics research is to make robots fully autonomous while they perform their duties. A robot should be able to cope with problems which could arise during its task. These are often unpredictable as a programmer cannot predict all of them in advance. Sources of errors may be due to inaccurate modelling of robot sensors, actuators, or environment. But problems often arise when the robot has to deal with a highly dynamic environment, like robotic museum tour-guides often do.

In this thesis we addressed the above problems by proposing fault tolerant robotic architectures. The first proposed architecture introduced a high level decision system to allow the robot taking decision when facing with obstacles to fullfill its task. The decision system works coupled with a dynamic Bayesian network to estimate unobservable world states. For example we tried to address the problem of people blocking the robot path. Depending on sensor data and on an estimate of the people number, the robot was able to decide how to react. He asked people to move aside when avoiding them was too hard, while it decided to outmaneuver them if there was enough space. This behavior was not provided in advance to the robot, but it emerged from the decision system. This example proved that a probabilistic reasoning system coupled with low-level routines increased the robot performances while operating in a dynamic environment.

We had to face different problems while dealing with an outdoor environment. Here large spaces and poor odometry made localization based on metric maps unreliable. The first proposed outdoor robotic system used GPS data to cluster the environment, and fixed route points to navigate. Although the system proved reliable, its main drawback is that it relies on human made environment representation, limiting robot autonomous capabilities. We tried to address this issue in the second proposed architecture, which employs appearance based place recognition. According to this approach, the robot is able to detect places in the environment using holistic features extraction from camera images. The data association problem is addressed by coupling sensed images with odometric information, which is reliable for small distances.

In the third and last robotic architecture we proposed a functional definition of meaningful places. In contrast with the previous approach, where places were detected according to differences in images and distances between them, a place is detected if it is useful for the robot task. As one of the main tasks is navigation, we defined a place as a subset of the environment where a branching is present. A topological map has been used to represent the environment, and a novel probabilistic algorithm has been proposed to update the map while the robot gains new information.

Future works include adapting the high level decision system to the outdoor robotic systems, mainly to decide where to explore in order to build and refine the topological map. The robot could also use the decision system in order to detect

nodes using its own notion of "functional". Furthermore, as a robotic museum tour-guide has to interact with people, we should introduce advanced human-robot interaction algorithms to make it more appealing. Finally we are working on providing the robot with a capability to understand spoken language in order to introduce an easier way to interact with it.

Bibliography

[1] I. Horswill, "Specialization of Perceptual Processes," 1995.

[2] W. Burgard, A. Cremers, D. Fox, D. Hahnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "The interactive museum tour-guide robot," *Proceedings of AAAI*, vol. 98, pp. 11–18, 1998.

[3] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt, "Map learning and high-speed navigation in RHINO," *Artificial intelligence and mobile robots: case studies of successful robot systems table of contents*, pp. 21–52, 1998.

[4] S. Koenig and R. Simmons, "Xavier: a robot navigation architecture based on partially observable Markov decision process models," *Artificial intelligence and mobile robots: case studies of successful robot systems table of contents*, pp. 91–122, 1998.

[5] J. Pineau and G. Gordon, "POMDP planning for robust robot control," *The Twelveth International Symposium on Robotics Research*, 2005.

[6] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, *et al.*, "Probabilistic Algorithms and the Interactive Museum Tour-Guide Robot Minerva," *The International Journal of Robotics Research*, vol. 19, no. 11, pp. 972–999, 2000.

[7] R. Siegwart, K. Arras, S. Bouabdallah, D. Burnier, G. Froidevaux, X. Greppin, B. Jensen, A. Lorotte, L. Mayor, M. Meisser, *et al.*, "Robox at Expo. 02: A large-scale installation of personal robots," *Robotics and Autonomous Systems*, vol. 42, pp. 203–222, 2003.

[8] I. Nourbakhsh, J. Bobenage, S. Grange, R. Lutz, R. Meyer, and A. Soto, "An affective mobile robot educator with a full-time job," *Artificial Intelligence*, vol. 114, no. 1-2, pp. 95–124, 1999.

[9] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[10] R. Kalman, "A new approach to linear filtering and prediction problems," *Trans. ASME, Journal of Basic Engineering*, no. 82, pp. 35–45, 1960.

[11] S. Julier and J. Uhlmann, "A new extention of the kalman filter to nonlinear systems," in *International Symposium on Aerospace/Defence Sensing, Simulate and Controls*, 1997.

[12] N. Metropolis and S. Ulam, "The monte carlo method," *Journal of the American Statistical Association*, no. 44, pp. 335–341, 1949.

[13] N. Gordon, D. Salmond, and A. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *Radar and Signal Processing, IEE Proceedings F*, vol. 140, no. 2, pp. 107–113, 1993.

[14] A. Doucet, "On Sequential Simulation-Based Methods for Bayesian Filtering," 1998.

[15] A. Doucet and N. De Freitas, *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[16] D. Fox, W. Burgard, and S. Thrun, "Markov localization for mobile robots in dynamic environments," *Journal of Artificial Intelligence Research*, vol. 11, pp. 391–427, 1999.

[17] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localization for mobile robots," *Artificial Intelligence*, 2001.

[18] S. Thrun, "Robotic Mapping: A Survey," *Exploring Artificial Intelligence in the New Millennium*, 2003.

[19] H. Moravec and A. Elfes, "High resolution maps from wide angular sensors," in *Proceedings of the IEEE International Conference On Robotics and Automation (ICRA-85)*, pp. 116–121, IEEE Press, 1985.

[20] H. Moravec and M. Martin, "Robot navigation by 3D spatial evidence grids," *Mobile Robot Laboratory, RoboticsInstitute, CarnegieMellonUniversity*, 1994.

[21] A. Dempster, N. Laird, and D. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[22] S. Thrun, W. Burgard, and D. Fox, "A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots," *Autonomous Robots*, vol. 5, no. 3, pp. 253–271, 1998.

[23] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *Robotics & Automation Magazine, IEEE*, vol. 4, no. 1, pp. 23–33, 1997.

[24] A. Madsen and U. Kjærulff, "Applications of HUGIN to Diagnosis and Control of Autonomous Vehicles," *STUDIES IN FUZZINESS AND SOFT COMPUTING*, vol. 213, p. 313, 2007.

[25] S. Lauritzen and D. Nilsson, "Representing and Solving Decision Problems with Limited Information," *Management Science*, vol. 47, no. 9, pp. 1235–1251, 2001.

[26] O. Lebeltel, P. Bessière, J. Diard, and E. Mazer, "Bayesian Robot Programming," *Autonomous Robots*, vol. 16, no. 1, pp. 49–79, 2004.

[27] G. Infantes, F. Ingrand, and M. Ghallab, "Learning Behaviors Models for Robot Execution Control," *Proc. 17th European Conference on Artificial Intelligence ECAI*, 2006.

[28] X. Perrin, R. Chavarriaga, R. Siegwart, and J. Millan, "Bayesian controller for a novel semi-autonomous navigation concept," in *Proceedings of the 3rd European Conference on Mobile Robots*, 2007.

[29] G. Theocharous, K. Murphy, and L. Kaelbling, "Representing hierarchical POMDPs as DBNs for multi-scale robot localization," *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 1, 2004.

[30] D. Fox, S. Thrun, W. Burgard, and F. Dellaert, "Particle filters for mobile robot localization," in *Sequential Monte Carlo Methods in Practice* (A. Doucet, N. de Freitas, and N. Gordon, eds.), pp. 499–516, Springer Verlag, 2001.

[31] M. Bosse, P. Newman, J. Leonard, M. Soika, W. Feiten, and S. Teller, "An Atlas framework for scalable mapping," *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 2, 2003.

[32] J. Blanco, J. Fernandez-Madrigal, and J. Gonzalez, "A New Approach for Large-Scale Localization and Mapping: Hybrid Metric-Topological SLAM," *Robotics and Automation, 2007 IEEE International Conference on*, pp. 2061–2067, 2007.

[33] C. Estrada, J. Neira, and J. Tardos, "Hierarchical SLAM: Real-Time Accurate Mapping of Large Environments," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 588–596, 2005.

[34] A. Chella, I. Macaluso, and L. Riano, "Automatic Landmark Detection and Recognition in Autonomous Robotics," in *Proceedings of the 2007 International Joint Conference on Artificial Intelligence (IJCAI), Workshop on Attention in Cognitive Systems*, 2007.

[35] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 1156, 2003.

[36] J. M. Porta and B. Kröse, "Appearance-based concurrent map building and localization," *Robotics and Autonomous Sustems*, vol. 54, 2006.

[37] Y. Matumoto, K. Sakai, M. Inaba, and H. Inoue, "View-based approach to robot navigation," in *Proceedings of the 2000 IEEE/RSJ International Intelligent Robots and Systems*, pp. 1702–1708, IEEE Press, 2000.

[38] S. Se, D. Lowe, and J. Little, "Mobile Robot Localization and Mapping with Uncertainty using Scale-Invariant Visual Landmarks," *The International Journal of Robotics Research*, vol. 21, no. 8, pp. 735–758, 2002.

[39] A. Chella and I. Macaluso, "Sensations and perceptions in cicerobot, a museum guide robot," in *BICS 2006 Conference*, 2006.

[40] A. Eliazar and R. Parr, "Learning probabilistic motion models for mobile robots," *ACM International Conference Proceeding Series*, 2004.

[41] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," *Robotics, IEEE Transactions on [see also Robotics and Automation, IEEE Transactions on]*, vol. 23, no. 1, pp. 34–46, 2007.

[42] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1999.

[43] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International Journal of Computer Vision*, vol. 42, no. 3, pp. 145–175, 2001.

[44] R. Sim and G. Dudek, "Comparing Image-based Localization Methods," in *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI), Acapulco, Mexico*, 2003.

[45] M. Jogan and A. Leonardis, "Robust Localization using Eigenspace of Spinning-Images," in *Proceedings of the IEEE Workshop on Omnidirectional Vision, Hilton Head Island, South Carolina*, 2000.

[46] B. Kröse, N. Vlassis, and R. Bunschoten, "Omnidirectional Vision for Appearance-based Robot Localization," in *Lecture Notes in Computer Science*, vol. 2238, Springer, 2002.

[47] H. V. Neto and U. Nehmzow, "Visual novelty detection for inspection tasks using mobile robots," in *Proceedings of the 8th Brazilian Symposium on Neural Networks (SBRN 2004)*, 2004.

[48] M. Artač, M. Jogan, and A. Leonardis, "Mobile Robot Localization Using an Incremental Eigenspace Model," in *Proceedings of the 2002 IEEE International Conference on Robotics Automation*, May 2002.

[49] A. Ranganathan, E. Menegatti, and F. Dellaert, "Bayesian inference in the space of topological maps," *IEEE Transactions on Robotics*, vol. 22, Feb. 2006.

[50] A. Torralba, K. Murphy, W. Freeman, and M. Rubin, "Context-based vision system for place and object recognition," in *Proceedings. Ninth IEEE International Conference on Computer Vision, 2003*, 2003.

[51] C. Siagian and L. Itti, "Rapid Biologically-Inspired Scene Classification Using Features Shared with Visual Attention," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 2, 2007.

[52] J. M. Porta, J. Verbeek, and B. Kröse, "Active Appearance-Based Robot Localization Using Stereo Vision," *Autonomous Robots*, vol. 18, no. 1, 2005.

[53] B. Yamauchi and R. Beer, "Spatial learning for navigation in dynamic environments," *IEEE Transactions on Systems, Man and Cybernetics-Part B.*, vol. 26, no. 3, pp. 496–505, 1996. Special Issue on Learning Autonomous Robots.

[54] K. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Boston: Academic Press, 1990.

[55] M. A. Figuerido and A. K. Jain, "Unsupurevised Learning of Finite Mixture Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 3, 2002.

[56] M. Song and H. Wan, "Highly Efficient Incremental Estimation of Gaussian Mixture Models for Online Data Stream Clustering," in *SPIE Conference on Intelligent Computing: Theory And Application III*, 2005.

[57] O. Ledoit and M. Wolf, "Some hypothesis tests for the covariance matrix when the dimension is large compared to the sample size," *The Annals of Statistic*, vol. 30, no. 4, 2002.

[58] A. Poncela, E. Perez, A. Bandera, C. Urdiales, and F. Sandoval, "Efficient integration of metric and topological maps for directed exploration of unknown environments," *Robotics and Autonomous Systems*, vol. 41, no. 1, pp. 21–39, 2002.

[59] B. Kuipers, "The Spatial Semantic Hierarchy," *Artificial Intelligence*, vol. 119, no. 1–2, pp. 191–233, 2000.

[60] S. Thrun, S. Thayer, W. Whittaker, C. Baker, W. Burgard, D. Ferguson, D. Hahnel, D. Montemerlo, A. Morris, Z. Omohundro, *et al.*, "Autonomous exploration and mapping of abandoned mines," *Robotics & Automation Magazine, IEEE*, vol. 11, no. 4, pp. 79–91, 2004.

[61] J. Leonard and H. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons," *Robotics and Automation, IEEE Transactions on*, vol. 7, no. 3, pp. 376–382, 1991.

[62] K. Murphy, "Bayesian map learning in dynamic environments," *Advances in Neural Information Processing Systems*, vol. 12, pp. 1015–1021, 2000.

[63] G. Grisetti, G. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi, "Fast and accurate SLAM with Rao-Blackwellized particle filters," *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 30–38, 2007.

[64] H. Shatkay and L. Kaelbling, "Learning Geometrically-Constrained Hidden Markov Models for Robot Navigation: Bridging the Topological-Geometrical Gap," *Journal of Artificial Intelligence Research*, vol. 16, pp. 167–207, 2002.

[65] N. Ayache, O. Faugeras, and L. INRIA, "Maintaining representations of the environment of a mobile robot," *Robotics and Automation, IEEE Transactions on*, vol. 5, no. 6, pp. 804–819, 1989.

[66] D. Lowe, "Object recognition from local scale-invariant features," *International Conference on Computer Vision*, vol. 2, pp. 1150–1157, 1999.

[67] S. Se, D. Lowe, and J. Little, "Vision-based mobile robot localization and mapping using scale-invariant features," *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 2, 2001.

[68] S. Se, D. Lowe, and J. Little, "Vision-based global localization and mapping for mobile robots," *Robotics, IEEE Transactions on [see also Robotics and Automation, IEEE Transactions on]*, vol. 21, no. 3, pp. 364–375, 2005.

[69] R. Sim, P. Elinas, M. Griffin, and J. Little, "Vision-based SLAM using the Rao-Blackwellised particle filter," *IJCAI Workshop on Reasoning with Uncertainty in Robotics*, 2005.

[70] A. Davison, "Real-time simultaneous localisation and mapping with a single camera," *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pp. 1403–1410, 2003.

[71] J. Shi and C. Tomasi, "Good features to track," *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pp. 593–600, 1994.

[72] M. Garcia and A. Solanas, "3D simultaneous localization and modeling from stereo vision," *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 1, 2004.

[73] C. Harris and M. Stephens, "A combined corner and edge detector," *Alvey Vision Conference*, vol. 15, p. 50, 1988.

[74] W. van der Mark, F. Groen, and J. van den Heuvel, "Stereo based navigation in unstructured environments," *Instrumentation and Measurement Technology Conference, 2001. IMTC 2001. Proceedings of the 18th IEEE*, vol. 3, 2001.

[75] P. Bellutta, R. Manduchi, L. Matthies, K. Owens, and A. Rankin, "Terrain perception for DEMO III," *Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE*, pp. 326–331, 2000.

[76] M. Lutzeler and E. Dickmanns, "EMS-vision: recognition of intersections on unmarked road networks," *Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE*, pp. 302–307, 2000.

[77] A. Chella, I. Macaluso, and L. Riano, "Automatic place detection and localization in autonomous robotics," *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2007.

[78] P. Newman, D. Cole, and K. Ho, "Outdoor SLAM using visual appearance and laser ranging," *IEEE International Conference on Robotics and Automation*, 2006.

[79] A. Erkan, R. Hadsell, P. Sermanet, J. Ben, U. Muller, and Y. LeCun, "Adaptive long range vision in unstructured terrain," in *Proc. Intelligent Robots and Systems (IROS'07)*, 2007.

[80] R. Haralick and L. Shapiro, *Computer and Robot Vision*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1992.

[81] T. Lozano-Perez, "Spatial Planning: A Configuration Space Approach," *IEEE Transactions on Computers*, vol. 32, no. 2, pp. 108–120, 1983.

[82] M. Context, "de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf," *Computational Geometry: Algorithms and Applications*, 1997.

[83] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[84] A. Chella, I. Macaluso, D. Peri, and L. Riano, "Robotanic: a robot guide for botanical gardens. early steps.," in *In Proceedings of the 6th AI*IA Conference, Cultural Heritage Workshop*, 2007.